Renesas Synergy^{TM}, RL78/G1D Healthcare Meters Kit

Modifications for Continua Certification

R30AN0335EU0100
Rev.1.00
Mar 15, 2018

Introduction

This document provides information about the modifications to the Bluetooth Low Energy (BLE) component of the Renesas SynergyTM, RL78/G1D Healthcare Meters Kit (HC Meters Kit). Built with the microcontrollers (MCUs) and software provided in the Renesas Synergy^{TM} Platform, these kits are certified to comply with the data interoperability standards defined by the Continua Health Alliance. These kits are reference designs which demonstrate key technologies available for developing interoperable telehealth devices for the three major categories: chronic disease management, aging independently, and health and physical fitness.

To achieve Continua Compliance, a medical device using BLE for communicating health data must meet the following requirements common to the Glucose meter, Blood Pressure Cuff, Pulse Oximeter, and Activity monitor:

1. Utilize a BLE SIG certified profile.
2. Provide Device Information Service to expose relevant product information.
3. Provide a readable and optionally (writable) Current Time Service.
4. Support BLE communication with Encryption and Man-in-the-Middle (MITM) protection schemes.
5. Use Medical Device Encoding Rules (MDER) when transferring data to a remote client collecting data from the device.

Compliance with these common requirements and other application specific requirements, as defined by test specifications, are demonstrated by the firmware accompanying this document. Blood Glucose Meter, Blood Pressure Cuff, and the Pulse Oximeter modules have been certified by Continua and listed on the Personal Connected Health Alliance (PCHA) website.

Target Device

RL78/G1D and R7FS3A77C3A01CFP

Contents

1. Development of the BTLE Adapter

The original Bluetooth Low Energy communications, provided by the RBLE stack operating in a modem configuration on the Synergy S3A7 and RL78/G1D MCU, consisted of a single custom service with a single measurement characteristic. The same service and characteristic was being repurposed to transfer health data irrespective of the type of medical device; like a Glucose meter, Blood Pressure Cuff, Pulse Oximeter, and Activity monitor.

The following modifications were made to the Healthcare Meters Kit to achieve data interoperability:

1. Move from RBLE stack version 1.10 to 1.20 to incorporate the latest available features.
2. Pairing and bonding is added, with either security-request or pairing-on-demand support. MITM (Man in the Middle) and Pairing-Just-Works are both supported.
3. To become Continua Compliant, the custom service is removed and replaced with the services and characteristics as specified in the Blood Pressure, Glucose, and Pulse Oximeter Service and Profile specifications, allowing measurements to be transferred with precision information.
4. A Device Information Service is added to support the product and Continua regulatory information as required by Continua.
5. A Current Time service is added to support the use of time stamps in the measurements and the Continua requirement of delivering such measurements to a remote destination with consistent time. Support for setting the current time by the collector is added with proper handling of date-time adjustments in the stored data.
6. Support is added for temporarily stored measurements in the Blood Pressure and Pulse Oximeter Adapters. The Glucose Adapter is designed to persistently store measurements taken offline and only deliver data using Record Access Control point transactions according to the profile specifications.
7. A basic Record Access Control point is supported in the Pulse Oximeter as required by the Pulse Oximeter service and profile specifications. The required feature characteristics are added for each Adapter type as specified by the certification.
8. Necessary information about the peer and what it has configured on the peripheral is persistently stored such that bonded reconnects occur according to the core specification.
9. The BTLE Adapter also supports unpaired connections if the vendor chooses to take that route. To the extent possible, the Adapter tracks actions such that it does not disconnect until a specified idle time is reached.
10. All measurement options, as noted in the profile specifications, are supported except the intermediary pressure in the Blood Pressure.

2. Demo Project

The Demo Project does not interface to any sensors. Instead, dummy data is generated at fixed intervals and sent to the adapter illustrating the possibilities allowed by the profile specifications (special values, error conditions, all the various types of measurement fields, and so on). The adapter handles them according to the connection state; sending the data if connected (except in the Glucose meter where data is not generated while connected) and storing them otherwise. For all but the pulse oximeter, the generation of streaming measurements is unrealistic, thus the demo disconnects after 5 seconds after sending its measurements. In most production cases, you will take just a single measurement and turn off the device.

3. Supported Profiles

The project supports three Continua compliant device specializations; the Blood Pressure Cuff, Pulse Oximeter, and Glucose Monitor. The default demonstration project is set up with the Glucose Monitor. **Although the Activity Monitor profile is provided, it is "expected" to be adopted or certified by the Continua Alliance.**

4. Required Support

The BTLE Adapter requires that the Synergy S3A7 Target Board running the medical application, be configured to utilize the Real-Time Clock for maintaining continuous time, and Flash Driver for allowing writes to the Data Flash to facilitate saving BLE peer pairing and host configuration information and stored measurements. The real-time clock circuit must be designed to allow continuous operation between power cycles.

5. BTLE Adapter Related Files

Since real data is not being generated for the scope of demonstration and testing, different certified profiles are emulated on the Blood Pressure module hardware. The files related to the BTLE Adapter and the RBLE HOST that allow the BTLE Adapter to communicate with the RL78/G1D are in the directory structure as shown below:

```
src
      application_dispatcher_thread_entry.c     thread that gets sensor data and
sends it to the BTLE
                                    Adapter
      ble_host_thread_entry.c       thread that initializes the BTLE Adapter and
starts the
                                    Bluetooth functionality
      RBLE
            src
                  btle_adapter
                        aes.c
                        aes.h
                        db_handle.h
                        glpProfileApp.c
                        glpProfileApp.h
                        glpProfileCommandHandler.c
                        glpProfileGAPCallbackHandlers.c
                        glpProfileGLPCallbackHandler.c
                        glpProfileSMCallbackHandlers.c
                        glpProfileUtilities.c
                        MderFloat.c
                        MderFloat.h
                        my_sleep.c
                        glucose          glucose profile files; excluded from
                  project
                        bloodPressure    blood pressure profile files; excluded
                  from project
                        pulseOximeter    pulse oximeter profile files; excluded
                  from project
                        activityMonitor activity monitor profile files; excluded
                  from project
                  host
                        gap
                        gatt
                        sm
                        vs
                        tip
                        blp   RBLE HOST support for the Blood Pressure Profile
                        glp   RBLE HOST support for the Glucose Profile
                        plxp        RBLE HOST support for the Pulse Oximeter
                  Profile
                  include
```

```
                    prf_sel.h  file that selects which profile is being
          supported
               rscip
```

The example above shows the project for the Glucose profile. The files and directories in blue are profile-specific. These files are replaced by the desired profile-specific files. The **glucose**, **bloodPressure**, **pulseOximeter,** and **activityMonitor** directories are just convenient storage locations containing the profile-specific files. It is important when building the project that the files in these directories are excluded from the build. Renesas Synergy™ e^2 studio will include the C-source files in these directories by default so you must take steps to assure that does not happen.

5.1 Changing Profiles

The application can change profiles by:

1. Copying the desired profile files in the `synergy_bpm\src\rBLE\src\btle_adapter\glucose`, synergy_bpm\src\rBLE\src\btle_adapter\bloodPressure, and `synergy_bpm\src\rBLE\src\btle_adapter\pulseOximeter` directories in the into the `synergy_bpm\src\rBLE\src\btle_adapter` directory, replacing the existing profile files in that directory. Each profile file begins with either `glp`, `blp`, or `plx` indicating that it belongs to the glucose, blood pressure, and pulse oximeter profile, respectively.
2. The `prf_sel.h` file found in the directories listed above is copied to the `synergy_bpm\src\rBLE\src\include directory`
3. The BTLE Adapter interface file `application_dispatcher_thread_entry.c` replaces the file of the same name in the `synergy_bpm\src` directory.
4. When using the activity monitor profile, it is important to use the `USE_SAMPLE_PROFILE` preprocessor in the compiler defined symbols. When using the other three profiles, this preprocessor may be changed to `noUSE_SAMPLE_PROFILE`.

It may simply be easier to select the required profiles in `prf_sel.h` instead of copying.

5.2 The BTLE Adapter and the RL78/G1D

The BTLE Adapter resides on the Synergy MCU and makes calls into the RL78/G1D MCU via the RBLE HOST interface. The intelligence of the profile sits in the BTLE Adapter. Nevertheless, there are some code modifications that every application developer will need to perform on the RL78/G1D MCU Target Board, for example, setting the Device Information Service (DIS) characteristic values to match the production device. The DIS is where one specifies the manufacturer name, model number, serial number, IEEE system identifier, firmware, hardware, and software versions, and more. The Bluetooth address is also set on the RL78/G1D MCU. Building the RL78/G1D HEX file can be done in e^2 studio, but it involves a separate compiler (CCRL), the Synergy Standalone Configurator (SSC) for intuitive configuration of microcontroller functions, software assignments, and operation modes.

6. RBLE HOST

The RBLE HOST is platform-dependent code that allows an application to make RL78/G1D API calls as if the code were being written on the RL78/G1D MCU. The platform-dependent part of the RBLE HOST lies mostly in that part of the code dedicated to communicating with the RL78/G1D MCU from the host MCU. In this manner the BTLE Adapter code provided for the Synergy can be used on any platform that has an RBLE HOST written for it with few modifications. Those modifications that are needed when moving to other platforms are for the methods that:

- obtain a current time
- perform a sleep
- save/restore data to/from non-volatile storage

The provided BTLE Adapter also runs on a PC. An RBLE HOST is provided for the PC in the RL78/G1D MCU distribution package. If the application developer is not interested in the PC option, the code can be simplified by removing the PC-specific conditional compilation statements. The pre-processor variable `PC_APP` identifies the PC-specific parts.

7. Running the Demo

Once the profile files are in place you can build the project in e^2 studio and download the HEX file onto the Synergy Target Board MCU. The `prf_sel.h` and `db_handle.h` files must be the same for both the RL78/G1D MCU and Synergy builds. Recall that the RL78/G1D MCU dongle must be loaded with the corresponding HEX file for the selected profile.

8. Using the BTLE Adapter in an Application

The BTLE Adapter requires minimal work on your part if the default configuration and behavior is sufficient. All the application needs to do is select some options, call the data initializer, start the sensor-data receive thread, and start the Bluetooth functionality. In the Demo Project this setup is done in the file `ble_host_thread_entry.c`. The application feeds data to the BTLE Adapter in the sensor-data receive thread by loading pre-defined data structures, and passing the data structures to the BTLE Adapter. The BTLE Adapter automatically stores data when not connected or unable to send, and sends the data otherwise. Bluetooth does not need to be started to handle sensor data; the application can pass sensor data to the BTLE Adapter when not connected and when the Bluetooth has not been started. In the demo project, the loading of the generated **sensor** data is done in the file `application_dispatcher_thread_entry.c.`

The demo default behavior is as follows:

8.1 Default Behavior

- On power-up:
  - The BTLE Adapter is configured for no Man in the Middle protection and no private addressing.
  - The BTLE Adapter data structures are initialized where any stored data in flash is read.
  - The measurement thread is started (measurements can now be taken).
  - Bluetooth functionality of the BTLE Adapter is started.
- The RL78/G1D and Synergy S3A7 MCU is initialized.
- After initialization, advertisements are sent with a friendly name and the 16 bit service UUID of the profile measurement service.
  - An application will likely want to change the friendly name.
  - An application may also want to control the start of advertising.
- The BTLE Adapter requires pairing to send measurements.
- Disconnects are sent `SECONDS_TO_DISCONNECT` seconds after the last detected `GATT` action after the measurements are sent.
  - In the case of the Pulse Oximeter if continuous measurements are enabled by the collector the demo does not disconnect unless you disconnect or the peer collector disconnects.
- Re-advertising starts automatically `SECONDS_TO_REBROADCAST` seconds after the disconnect.
  - An application is likely to change this behavior to shut down.
- You press the S4 switch to shut down the demo.
  - Device data is stored in flash.
  - Profile and measurement data are stored in flash.

8.2 Starting the BTLE Adapter

In the demo application the BTLE Adapter is initialized and started in the `ble_host_thread_entry.c` file in the **`synergy_bpm\src`** directory. Prior to starting the adapter, the application sets a few features it desires. Then it calls a method `APP_InitializeData()` that initializes the data structures including the reading of any persisted information. This method must be called before feeding any measurement data to the BTLE Adapter. Following the initialization, the thread that interfaces the sensor data to the BTLE Adapter is called. It is not necessary to start the Bluetooth functionality before running this thread as it is designed to handle measurements taken off line.

The Bluetooth functionality is started by calling `APP_Init()` and entering the so-called infinite embedded loop. No Bluetooth exchanges are possible before calling `APP_Init()`. In the demo configuration, the Bluetooth is started on power up. The application may want to delay the start of the Bluetooth until invoked by you or upon receiving a measurement from the sensor.

8.2.1 BTLE Adapter Methods

The BTLE Adapter provides a few simple methods for basic configuration, start up, and shutdown.

(1) **void registerStatusCallback(fnStatusCallback statusCallback, void *happ)**

This method provides the application with a means to receive status messages from the BTLE Adapter. The provided adapter has only a few status events defined; it is intended that the application will define its own set of status events and modify the BTLE Adapter to report the status messages the application is interested in. `fnStatusCallback statusCallback` points to the application-implemented callback function handling status messages. void `*happ` is an arbitrary pointer defined by the application. Whatever the application passes in is returned in when the callback is signaled.

(2) **void setUseMITM(BOOL set)**

This method indicates whether the pairing exchange should use **Man in the Middle** (MITM) protection or not. Setting this value to TRUE does not guarantee the use of MITM. Both sides must support MITM for the feature to be used. When used, the pairing process will require the use of a passkey of some type by you. The application must support a User Interface that can handle either the display of or entry of the passkey.

(3) **void setUseSecurityRequest(BOOL set)**

When TRUE the BTLE Adapter will send a security request to the peer on connection. If not paired, this request will cause the peer to initiate pairing. If already paired, this method will cause the peer to initiate encryption. If false, pairing will be done whenever the peer requests it or when the peer tries to do something that requires pairing, for example, setting the descriptor of measurement characteristics to enable transfers of data.

(4) **BOOL isConnected(void)**

Returns TRUE if the BTLE Adapter is connected to a peer collector.

(5) **void stopAdvertising(void)**

This method stops the sending of advertisements.

(6) **void APP_InitializeData(void)**

This method needs to be called before any data is delivered to the BTLE Adapter and the Bluetooth functionality is started. The primary task of the method is to read any persisted data from flash into the BTLE Adapter.

(7) **BOOL APP_Init(void)**

This method starts the Bluetooth Low Energy functionality. If the method succeeds, it returns TRUE. Calling it begins the cycle of `rBLE` calls into the RL78/G1D MCU followed by asynchronous responses. As soon as this call is made, the application needs to start the **infinite embedded loop** where `APP_Run()` and `rBLE_Run()` are called in a while loop that is only interrupted by calling the `btleAdapterShutdown()` method.

(8) **BOOL APP_Run(void)**

This method calls into the BTLE Adapter and for the handling of queued commands. It returns TRUE when the adapter has been shut down by calling `btleAdapterShutDown()`. The `APP_Run()` method must return in a timely manner that must be followed by the RBLE HOST call `rBLE_Run()`. All RBLE HOST implementations work in this manner. Note that the `rBLE_Run()`method is part of the RBLE HOST and not the BTLE Adapter.

(9) **void btleAdapterShutDown(void)**

This method terminates the BTLE Adapter. It is meant to be called when the user intends to turn off the device. The method disconnects from the peer if connected and saves the current peer collector information and any undelivered stored measurements to flash memory and then terminates the embedded infinite loop. Once this method is called, the BTLE Adapter can only be restarted by a hardware reset or power cycle.

8.3 Interfacing Sensor Data to the BTLE Adapter

To send measurement data the application populates one or more structures defined in the `*ProfileApp.h` file and passes it to the BTLE Adapter. Consequently, the primary effort with respect to Bluetooth on the part of the application is mapping the sensor data, in whatever form it may have, to the designated structures. The BTLE adapter will either store the data, discard it (only in the case of the Glucose meter when connected), or send it to the peer depending upon the rules specified in the profile. The details for populating the structures and how these structures are handled for each profile are given in the following sub sections. Helper methods are provided to assist in properly populating some of the structural elements.

8.3.1 IEEE 11073 MderFloats

Many of the measurement values in the structs are reported as IEEE 11073 MderFloats. MderFloats are used as this format provides a standardized means of reporting the precision of the value. In the MderFloat world, there is a difference between 2, 2.0, and 2.00. Numerically they are equal, but the 2.00 indicates the value is known out to a precision of two decimal places. An MderFloat has an exponent and a mantissa, both of which can be signed. To create a MderFloat Struct, a value is represented in the form $(\mathit{mantissa})*10^{\mathit{exponent}}$. Integers will have exponents greater than or equal to zero, and decimals will have negative exponents. The exponent indicates where in the mantissa the decimal point is placed. The following table illustrates the cases:

Mantissa	Exponent	Value	MDER_SFLOAT range
2	1	20	-2046 to 2045
-2	0	-2	-2046 to 2045
20	-1	2.0	-2046 to 2045
-200	-2	-2.00	-2046 to 2045
118	0	118	-2046 to 2045
1182	-1	118.2	-2046 to 2045
1182	-5	.01182	-2046 to 2045

The application will need to know what the precision of the sensor is. A support method

```
createMderFloat(sMderFloat *mder, short int exponent, long int mantissa, enum
MderFloatType mderType)
```

is provided that creates the MderFloat needed.

The application passes in the sMderStruct that needs to be populated, provides the exponent and mantissa, and selects the type of MderFloat. There are two types, `MDER_SFLOAT`, and `MDER_FLOAT`. Which type is used is determined by the Bluetooth specialization profile. The `MDER_SFLOAT` is only 16 bits with a 4 bit exponent and 12 bit mantissa, while the `MDER_FLOAT` is 32 bits with an 8 bit exponent and 24 bit mantissa. Clearly the `MDER_FLOAT` is used when more significant figures over larger ranges are needed. All the profiles discussed in this document use `MDER_SFLOATs`. The range of the mantissa for `MDER_SFLOAT` is from -2046 to 2045 if you have exponent 0. If you does the math one sees that the range should be -2048 to 2047, but there a set of special values, such as Not a Number, that are encoded in the upper ranges when the exponent is zero. You need to be aware of that limitation. On the other hand, if you need those values, you should probably be using `MDER_FLOATs` instead. For the supported specializations, the designers of the specifications have concluded that such values are not ever needed and `MDER_SFLOATs` are sufficient.

For the Blood Pressure, Glucose, and Pulse Oximeter profiles most values are reported as integers as measurements are not made with decimal precision. In that case the exponent is usually 0 and the mantissa is the value. Decimal precision is more commonly used when measurements are taken in mmol or hPa and in the case of the Pulseatile Quality. In the end, the precision reported depends upon the sensor specifications.

(1) MderFloat Special Values

`MderFloats` also provide a means of indicating special values, such as Not a Number, NaN, positive and negative infinity, not at this resolution, and reserved. A support method;

```
createMderSpecialValue(sMderFloat *mder, enum MderFloatType mderType, enum
MderSpecialValue specialValue);
```

is provided that creates the special value. The enumeration codes for the special values are one of:

```
NUMBER  // Normal number defined as 1 so be sure to set something!
NAN     // Not a Number
PINF    // Positive infinity
NINF    // Negative infinity
NRES    // Not at this resolution
RSVD    // Reserved
```

Applications will probably find the NAN and perhaps the PINF and NINF useful.

8.3.2 Blood Pressure Cuff

In the case of the Blood Pressure Profile, the data struct loaded is `sBpMeasurement`. Once loaded with data, the application passes the data to the `APP_Send_BP_Data(sBpMeasurement *)` method. If a peer is connected, paired, and enabled, the BTLE Adapter will send the data to the peer. If the BTLE Adapter is unable to deliver the measurement, the data will be stored in a buffer of size `NUMBER_OF_TEMP_BP_STORED_MEASUREMENTS` defined in `blpProfileApp.h.` When full, the oldest data is replaced by the newest. Stored data is sent and deleted as soon as the peer is ready to receive data. The value of `NUMBER_OF_TEMP_BP_STORED_MEASUREMENTS` is configurable.

(1) The sBpMeasurement Struct

The Blood Pressure profile does not have too many options. It must report the Blood Pressure measurement, may report a pulse rate, may report device error statuses, may use the simple person identifier, and may use a time stamp. If data is stored, reporting a time stamp is mandatory for Continua compliance. Thus, measurements without a time stamp will not be stored by the BTLE Adapter. The units of the Blood Pressure are either kilopascals, kPa, or millimeters of Mercury, mmHg. Which units are used is given by a flag. The details for populating the struct are given below:

```
typedef struct BP_MEASUREMENT_t
{
    BOOL use_kPas;     /** Set to TRUE if the Blood Pressure units are in kPa (kilo-Pascals).
                        * Otherwise the units are mmHg */
    BOOL hasTimeStamp; /** Set to TRUE if the measurement has a time stamp. MUST have a time stamp if
                        * the measurement is to be stored. */
    BOOL hasPulseRate; /** Set to TRUE if the measurement has a pulse rate */
    BOOL hasPersonId;  /** Set to TRUE if the measurement has a person id */
    BOOL hasStatus;    /** Set to TRUE if the measurement has a status field */

    uint8_t personId;  /** The person id: a value from 0 to 0xFE; a value of 0xFF means unknown
                        * person. The person id is just a number and the association of the number
                        * to a patient is made by the collector. */
    unsigned short status;    /** The measurement status: One or more of
                                      BP_BODY_MOVEMENT_DETECTED
                                              BP_CUFF_TOO_LOOSE
                                      BP_IRREGULAR_PULSE_DETECTED
                                      BP_PULSE_RATE_ABOVE_LIMIT
                                      BP_PULSE_RATE_BELOW_LIMIT
                                      BP_BAD_MEASUREMENT_POSITION
                              * Select multiple cases by using OR, eg: ->status =
                              *       (CUFF_TOO_LOOSE | BODY_MOVEMENT_DETECTED);
                              * One must be sure that the Blood Pressure feature value indicates
                              * support for the above conditions for the GATT Client to report
                              * the matching status settings. In other words, the Blood Pressure
                              * Feature must indicate support for BP_CUFF_TOO_LOOSE if one reports
                              * that status event. If no support is indicated, sending the event
                              * BP_CUFF_TOO_LOOSE will be ignored. The Blood Pressure feature
                              * requires coding and building the RL78 HEX file. The
                              * current RL78 project for the BP is set to 0x0F which indicates
                                      * support for all the above status conditions. */
    sMderFloat systolic;              /**  The  systolic  blood  pressure  as  an  MDER_SFLOAT.  See
MderFloat.h */
    sMderFloat diastolic;     /** The diastolic blood pressure as an MDER_SFLOAT. See MderFloat.h */
    sMderFloat mean;          /** The MAP/mean blood pressure as an MDER_SFLOAT. See MderFloat.h */
    sMderFloat pulseRate;     /** The pulse rate in Beats per minute as an MDER_SFLOAT.
                                * See MderFloat.h
                                * For the Blood Pressure cuff, these measurements are typically
                                * reported as integers. Thus the exponent will be 0 and the mantissa
                                * will be the integer value. */
    RBLE_DATE_TIME timestamp; /** The time stamp of the measurement. The structure contains
                                * uint16_t year      for example, 2017
                                * uint8_t month      1 - 12
                                * uint8_t day                1 – last day of month
                                * uint8_t hour               0 - 23
                                * uint8_t minute     0 - 59
                                * uint8_t second     0 - 59
                                * The BTLE Adapter provides a method getCurrentTmRtcTime(struct tm *)
                                * to get the Synergy current time and a second method
                                * BOOL tmToRbleBtleTime(struct tm *, RBLE_DATE_TIME *) to convert the
                                * tm struct to the RBLE_DATE_TIME struct. */
    time_t unixTimeStamp;     /** The equivalent time stamp as a unix EPOCH (seconds since 1970). We
                                * will use this to help correct stored time stamps on a set time. The
                                *  BTLE  Adapter  has  a  method  getCurrentUnixRtcTime(time_t  *)  to  get
                      the
                                * current EPOCH from the Synergy. */
}sBpMeasurement;
```

8.3.3 Pulse Oximeter

In the case of the Pulse Oximeter Profile, there are two data structures, one for so-called stable spot modality measurements, `sPoSpotMeasurement`, and one for continuous measurements, sPoContMeasurement. Once loaded with data, the application passes the data to the `APP_Send_PO_Spot_Data(sPoSpotMeasurement *)` and/or `APP_Send_PO_Cont_Data(sPoContMeasurement *)` methods. If a peer is connected, paired, and enabled, the BTLE Adapter will send the data to the peer. If it cannot send the data to the peer, the SPOT data will be stored in a buffer of size `NUMBER_OF_TEMP_PO_STORED_MEASUREMENTS` defined in `plxProfileApp.h.` When full, the oldest data is replaced by the newest. However, unlike the Blood Pressure case, stored data is not sent unless the peer asks for it through a so-called Record Access Control Point (RACP) transaction. Deletion of the data may also occur though an RACP transaction. However, the application is free to block or control its stored data as it sees fit. The value of `NUMBER_OF_TEMP_PO_STORED_MEASUREMENTS` is configurable. Continuous data is not saved, and if there is no connection, the measurement is dropped.

(1) The sPoSpotMeasurement Struct

The spot measurement is the simpler of the two structs. Spot measurements are another way of saying stable average. The sensor usually takes a while before it can obtain such a measurement. In the pulse oximeter profile, only spot measurements can be stored. If they are stored, a time stamp is mandatory for Continua compliance thus measurements without a time stamp will not be stored by the BTLE Adapter.

```
typedef struct PO_SPOT_MEASUREMENT_t
{
        BOOL hasTimeStamp;              /* Set to TRUE if the measurement has a timestamp. If the
                                         * measurement is stored, it shall have a timestamp. The
                                         * timestamp is reported in the ‘timestamp’ field. */
        BOOL hasMeasurementStatus;      /* Set to TRUE if one is reporting a measurement status.
                                         * The status itself is reported in the ‘measurementStatus’
                                         * field. It may carry multiple settings.
        BOOL hasDeviceAndSensorStatus;  /* Set to TRUE if a device or sensor status condition is to
                                         * be reported. The actual status conditions are reported in
                                         * the ‘deviceAndSensorStatus’ field.
        BOOL hasPulseAmplitude;         /* Set to TRUE if the measurement also contains a pulse signal
                                         * quality measure. The value is in percent. */
        BOOL needsTimeSet;              /* Set to TRUE if the device needs its time set by the peer. */
        unsigned short measurementStatus;
                /** The measurement status: One or more of
                        RBLE_PLXP_MEAS_STS_MEASUREMENT_ONGOING           Measurement Ongoing
                        RBLE_PLXP_MEAS_STS_EARLY_ESTIMATED_DATA          Early Estimated Data
                        RBLE_PLXP_MEAS_STS_VALIDATED_DATA                Validated Data
                        RBLE_PLXP_MEAS_STS_FULLY_QUALIFIED_DATA          Fully Qualified Data
                        RBLE_PLXP_MEAS_STS_DATA_FROM_MEASUREMENT_STORAGE Data from Measurement Storage
                        RBLE_PLXP_MEAS_STS_DATA_FOR_DEMONSTRATION        Data for Demonstration
                        RBLE_PLXP_MEAS_STS_DATA_FOR_TESTING              Data for Testing
                        RBLE_PLXP_MEAS_STS_CALIBRATION_ONGOING           Calibration Ongoing
                        RBLE_PLXP_MEAS_STS_MEASUREMENT_UNAVAILABLE       Measurement Unavailable
                        RBLE_PLXP_MEAS_STS_QUESTIONABLE_MEASUREMENT_DETECTED      Questionable Measurement
                                                                                  Detected
                        RBLE_PLXP_MEAS_STS_INVALID_MEASUREMENT_DETECTED  Invalid Measurement Detected
                  * Select multiple cases by using OR for example,
                  *     (RBLE_PLXP_MEAS_STS_MEASUREMENT_ONGOING | RBLE_PLXP_MEAS_STS_DATA_FOR_TESTING)
                  * One must be sure that the Pulse Ox feature value indicates support for the above
                  * conditions in order for the GATT Client to report the matching status settings.
                  * The Pulse Ox feature, unfortunately, requires coding and building the
                  * RL78 HEX file. The current RL78 project for the PulseOx supports all the above */
        unsigned long deviceAndSensorStatus;
                /** The device and sensor status: One or more of
                        RBLE_PLXP_DEV_STS_EXTENDED_DISPLAY_UPDATE_ONGOING        Extended Display Update
                                                                                 Ongoing
                        RBLE_PLXP_DEV_STS_EQUIPMENT_MALFUNCTION_DETECTED         Equipment Malfunction
                                                                                 detected
                        RBLE_PLXP_DEV_STS_SIGNAL_PROCESSING_IRREGULARITY_DETECTED Signal Processing
                                                                                  Irregularity Detected
                        RBLE_PLXP_DEV_STS_INADEQUATE_SIGNAL_DETECTED        Inadequate Signal Detected
                        RBLE_PLXP_DEV_STS_POOR_SIGNAL_DETECTED              Poor Signal Detected
                        RBLE_PLXP_DEV_STS_LOW_PERFUSION_DETECTED            Low Perfusion Detected
                        RBLE_PLXP_DEV_STS_ERRATIC_SIGNAL_DETECTED           Erratic Signal Detected
                        RBLE_PLXP_DEV_STS_NON_PULSATILE_SIGNAL_DETECTED     Non-Pulsatile Signal Detected
                        RBLE_PLXP_DEV_STS_QUESTIONABLE_PULSE_DETECTED       Questionable Pulse Detected
                        RBLE_PLXP_DEV_STS_SIGNAL_ANALYSIS_ONGOING           Signal Analysis Ongoing
                        RBLE_PLXP_DEV_STS_SENSOR_INTERFERENCE_DETECTED      Sensor Interference Detected
                        RBLE_PLXP_DEV_STS_SENSOR_UNCONNECTED_TO_USER        Sensor Unconnected to User
                        RBLE_PLXP_DEV_STS_UNKNOWN_SENSOR_CONNECTED          Unknown Sensor Connected
                        RBLE_PLXP_DEV_STS_SENSOR_DISPLACED                  Sensor Displaced
                        RBLE_PLXP_DEV_STS_SENSOR_MALFUNCTIONING             Sensor Malfunctioning
                        RBLE_PLXP_DEV_STS_SENSOR_DISCONNECTED               Sensor Disconnected

                  * Select multiple cases by using OR for example,
                  *   (RBLE_PLXP_DEV_STS_POOR_SIGNAL_DETECTED | BLE_PLXP_DEV_STS_ERRATIC_SIGNAL_DETECTED)
                  * One must be sure that the Pulse Ox feature value indicates support for the above
                  * conditions for the GATT Client to report the matching status settings.
```

```
              * The Pulse Ox feature requires coding and building the RL78 HEX file. The current
              * RL78 project for the PulseOx supports all the above */
    sMderFloat spO2;                /* The oxygen saturation as an MDER_SFLOAT; units percent */
    sMderFloat pulseRate;           /* The pulse rate as an MDER_SFLOAT; units beats per minute */
    sMderFloat pulseAmplitude;      /* The pulsatile quality as an MDER_SFLOAT; units percent */
    RBLE_DATE_TIME timestamp;       /* The time stamp. Structure shown above. */
    time_t unixTimeStamp;    /** We will use this to help correct time stamps on a set time */
}sPoSpotMeasurement;
```

(2) The sPoContMeasurement Struct

The continuous measurement supports the reporting of measurements taken using fast, slow, and **no** modalities. A single pulsatile quality may also be reported. If a continuous measurement is present, the **no** modality must be present. All three may be present. The continuous measurement has no timestamp and is sent unacknowledged. Since the measurement is delivered at some frequent interval like every one or two seconds, the time of reception is considered sufficiently accurate.

```
typedef struct PO_CONT_MEASUREMENT_t
{
    BOOL hasMeasurementStatus;                      /* Set to TRUE if the measurement reports a status
*/
    BOOL hasDeviceAndSensorStatus;           /* Set to TRUE if the measurement reports a device or
                                              * sensor status. */
    BOOL hasPulseAmplitude;                  /* Set to TRUE if the measurement contains a pulsatile
                                              * quality */
    BOOL hasFastModality;                    /* Set to TRUE if the measurement reports a FAST modality
                                              * oxygen saturation and pulse rate. */
    BOOL hasSlowModality;                    /* Set to TRUE if the measurement reports a SLOW modality
                                              * oxygen saturation and pulse rate. */
    unsigned short measurementStatus;        /* Measurement status options. One of more of
        /*  RBLE_PLXP_MEAS_STS_MEASUREMENT_ONGOING      Measurement Ongoing
            RBLE_PLXP_MEAS_STS_EARLY_ESTIMATED_DATA     Early Estimated Data
            RBLE_PLXP_MEAS_STS_VALIDATED_DATA           Validated Data
            RBLE_PLXP_MEAS_STS_FULLY_QUALIFIED_DATA     Fully Qualified Data
            RBLE_PLXP_MEAS_STS_DATA_FROM_MEASUREMENT_STORAGE  Data from Measurement Storage
            RBLE_PLXP_MEAS_STS_DATA_FOR_DEMONSTRATION   Data for Demonstration
            RBLE_PLXP_MEAS_STS_DATA_FOR_TESTING         Data for Testing
            RBLE_PLXP_MEAS_STS_CALIBRATION_ONGOING      Calibration Ongoing
            RBLE_PLXP_MEAS_STS_MEASUREMENT_UNAVAILABLE  Measurement Unavailable
            RBLE_PLXP_MEAS_STS_QUESTIONABLE_MEASUREMENT_DETECTED Questionable Measurement Detected
            RBLE_PLXP_MEAS_STS_INVALID_MEASUREMENT_DETECTED  Invalid Measurement Detected
        * Select multiple cases by using OR for example,
        *     (RBLE_PLXP_MEAS_STS_MEASUREMENT_ONGOING | RBLE_PLXP_MEAS_STS_DATA_FOR_TESTING)
        * One must be sure that the Pulse Ox feature value indicates support for the above
        * conditions for the GATT Client to report the matching status settings.
        * The Pulse Ox feature, unfortunately, requires coding and building the
        * RL78 HEX file. The current RL78 project for the PulseOx supports all the above */
    unsigned long deviceAndSensorStatus;
        /** The device and sensor status: One or more of
            RBLE_PLXP_DEV_STS_EXTENDED_DISPLAY_UPDATE_ONGOING   Extended Display Update Ongoing
            RBLE_PLXP_DEV_STS_EQUIPMENT_MALFUNCTION_DETECTED    Equipment Malfunction Detected
            RBLE_PLXP_DEV_STS_SIGNAL_PROCESSING_IRREGULARITY_DETECTED Signal Processing Irregularity
                                                              Detected
            RBLE_PLXP_DEV_STS_INADEQUATE_SIGNAL_DETECTED        Inadequate Signal Detected
            RBLE_PLXP_DEV_STS_POOR_SIGNAL_DETECTED              Poor Signal Detected
            RBLE_PLXP_DEV_STS_LOW_PERFUSION_DETECTED            Low Perfusion Detected
            RBLE_PLXP_DEV_STS_ERRATIC_SIGNAL_DETECTED           Erratic Signal Detected
            RBLE_PLXP_DEV_STS_NON_PULSATILE_SIGNAL_DETECTED     Non-Pulsatile Signal Detected
            RBLE_PLXP_DEV_STS_QUESTIONABLE_PULSE_DETECTED       Questionable Pulse Detected
            RBLE_PLXP_DEV_STS_SIGNAL_ANALYSIS_ONGOING           Signal Analysis Ongoing
            RBLE_PLXP_DEV_STS_SENSOR_INTERFERENCE_DETECTED      Sensor Interference Detected
            RBLE_PLXP_DEV_STS_SENSOR_UNCONNECTED_TO_USER        Sensor Unconnected to User
            RBLE_PLXP_DEV_STS_UNKNOWN_SENSOR_CONNECTED          Unknown Sensor Connected
            RBLE_PLXP_DEV_STS_SENSOR_DISPLACED                  Sensor Displaced
            RBLE_PLXP_DEV_STS_SENSOR_MALFUNCTIONING             Sensor Malfunctioning
            RBLE_PLXP_DEV_STS_SENSOR_DISCONNECTED               Sensor Disconnected

        * Select multiple cases by using OR for example,
        *     (RBLE_PLXP_DEV_STS_POOR_SIGNAL_DETECTED | RBLE_PLXP_DEV_STS_ERRATIC_SIGNAL_DETECTED)
        * One must be sure that the Pulse Ox feature value indicates support for the above
        * conditions for the GATT Client to report the matching status settings.
        * The Pulse Ox feature, unfortunately, requires coding and building the
        * RL78 HEX file. The current RL78 project for the PulseOx supports all the above */
    sMderFloat spO2;          /* The oxygen saturation as an MDER_SFLOAT; units percent */
    sMderFloat pulseRate;     /* The pulse rate as an MDER_SFLOAT; units bpm */
    sMderFloat spO2Fast;      /* The FAST modality oxygen saturation as an MDER_SFLOAT; units % */
    sMderFloat pulseRateFast; /* The FAST modality pulse rate as an MDER_SFLOAT; units bpm */
    sMderFloat spO2Slow;              /* The SLOW modality oxygen saturation as an MDER_SFLOAT; units %
*/
    sMderFloat pulseRateSlow;         /* The SLOW modality pulse rate as an MDER_SFLOAT; units bpm */
    sMderFloat pulseAmplitude;        /* The pulsatile quality as an MDER_SFLOAT; units per cent */
}sPoContMeasurement;
```

8.3.4 Glucose Monitor

In the case of the Glucose Profile, the data struct loaded by the application is `sGlMeasurement`. Once loaded with data, the application passes the data to the `APP_Load_GL_Data(sGlMeasurement *)` method. The Glucose monitor is special in the sense data is never sent live. If the monitor is connected to a peer, the method will simply return doing nothing with the measurement. When not connected, the measurement is stored and will be uploaded when connected to a peer and the peer requests it. The peer requests it through a so-called Record Access Control Point (RACP) transaction as in the Pulse Oximeter. However, the Glucose Profile supports many more RACP transactions than the Pulse Oximeter profile, including partial uploads and partial deletes. An application does not have to support delete transactions.

The Glucose buffer stores up to `NUMBER_OF_GL_STORED_MEASUREMENTS` measurements. The value is configurable. Each measurement takes 44 bytes of storage. When the buffer is full, the oldest message is deleted and the newest one added. Applications should warn you when the buffer is getting full to assure that they upload and delete old data so no measurements are lost.

(1) **sGlMeasurement Struct**

The Glucometer profile is relatively complex as it supports several contextual measurements such as the most recent meal, exercise, medication, health, and more, associated with the measurement in addition to the basic concentration. It is also possible to send a measurement that does not contain a concentration as might be the case when sending an `HbA1C` value. The application needs to populate this struct in a consistent manner for what it wants to report.

```
typedef struct GL_MEASUREMENT_t
{
        BOOL hasOffset;                  /* This field is set to TRUE if the user changed the time. The
                                 * amount of change is reported in the ‘offset’ field in minutes */
        BOOL hasConcTypeLoc;   /* This field is set to TRUE if a glucose concentration measurement
                                 * is to be reported, which is most cases. Note that if one reports
                                 * a concentration a location (where on the body it was taken) and
                                 * type (plasma, whole blood, etc.) must be specified. These fields
                                 * are usually determined by the meter type and do not change.
                                 * A special case where one might send a measurement without sending
                                 * a concentration would be if one is sending an hbA1C value. The
                                 * concentration, type, and location are reported in the
                                 * ‘glucConcentration’, ‘sourceType’ and ‘location’ fields */
        BOOL hasStatus;                  /* This field is set to TRUE if one wants to report a device or
sensor
                                 * status condition; typically an error. Again, there is a set of
                                 * possible codes one uses to report the actual status. Note that if
                                 * one is going to report status errors, that the Glucose Feature
                                 * characteristic on the RL78 must be set accordingly to indicate that
it
                                 * supports the said status error. The feature has a setting for each
                                 * possible status error. If the feature is not set, the setting in the
                                 * status field will be ignored by the peer. Reported in the ‘status’
*/
        BOOL hasMeal;          /* This field is set to TRUE if one is to report a meal context. The
meal
                                 * context is one of a few possible values typically selected by the
user
                                 * after taking a measurement. Reported in the ‘meal’ field. */
        BOOL hasCarbohydrates; /* This field is set to TRUE if the amount of carbohydrates consumed is
                                 * to be reported. The amount is in grams and reported in
                                 * the ‘carbohydateAmount’ field. When this field is reported one must
                                 * also report a ‘carbohydrateId’ value. */
        BOOL hasTesterHealth;  /* This field is set to TRUE is one is reporting general health status.
                                 * When a health status is reported one must also report a tester value
                                 * which in most cases is the user (SELF). These values are reported in
                                 * the ‘tester’ and ‘health’ fields.
        BOOL hasExercise;      /* This field is set to TRUE if one is reporting any exercise context.
                                 * Both the intensity and the duration of the exercise are reported in
                                 * the ‘intensity’ and ‘duration’ fields, respectively.
        BOOL hasMedication;    /* This field is set to TRUE if one is reporting the medication taken
                                 * context. The medication type is reported in the ‘medicationId’ field
                                 * and the amount is reported in the ‘medicationId’ field. The units of
                                 * the amount are given by the setting of the ‘hasMedicationLitersUnits’
                                 * boolean. When TRUE, the units are ml, when FALSE, mg.
        BOOL hasMedicationLitersUnits; /* Set to TRUE if the medication units are in milliliters.
        BOOL hasHbA1C;          /* Set to TRUE if reporting an HbA1C value. Reported in the 'hbA1C'
field
                                 * in percent.
        BOOL hasContext;                /* This field does not need to be set by the application */
        BOOL hasmmol_dLUnits;   /* Set to FALSE if the glucose concentration is provided in mg/dL
                                 * TRUE indicates concentration is mmol/L. The measurement
                                 * values will be converted to the ridiculous units used
                                 * by the Glucose profile/service specification. */
        BOOL medUnits;          /* If TRUE the medication units are milliliters; If FALSE = milligrams
*/
========================================          FIELDS           TO           BE           FILLED
===========================================
        uint8_t sourceType;     /* 'Blood' types from which the measurement is made. One and only one
of
                                        RBLE_GLP_TYPE_CAPILLARY_WHOLE_BLOOD
                                        RBLE_GLP_TYPE_CAPILLARY_PLASMA,
                                        RBLE_GLP_TYPE_VENOUS_WHOLE_BLOOD,
                                        RBLE_GLP_TYPE_VENOUS_PLASMA,
                                        RBLE_GLP_TYPE_ARTERIAL_WHOLE_BLOOD,
                                        RBLE_GLP_TYPE_ARTERIAL_PLASMA,
                                        RBLE_GLP_TYPE_UNDETERMINED_WHOLE_BLOOD,
                                        RBLE_GLP_TYPE_UNDETERMINED_PLASMA,
                                        RBLE_GLP_TYPE_ISF,
                                        RBLE_GLP_TYPE_CONTROL_SOLUTION */
        uint8_t location;       /* One and only one of
                                        RBLE_GLP_SAMPLELOC_FINGER,
                                        RBLE_GLP_SAMPLELOC_AST,             (Alternate site test)
                                        RBLE_GLP_SAMPLELOC_EARLOBE,
                                        RBLE_GLP_SAMPLELOC_CONTROL_SOLUTION,
                                        RBLE_GLP_SAMPLELOC_NOT_AVALABLE */
        short int offset;               /* Time offset adjustment in minutes */
        unsigned short sensorStatus;  /** The sensor status annunciation: One or more of
                RBLE_GLP_SENSORSTATUS_DEVICE_BATTERY_LOW    Device battery low at time of measurement
                RBLE_GLP_SENSORSTATUS_MALFUNCTION_FAULTING  Sensor malfunction or faulting at time of
                                                           measurement
                RBLE_GLP_SENSORSTATUS_INSUFFICIENT             Sample size for blood or control solution
                                                           Insufficient at time of measurement
                RBLE_GLP_SENSORSTATUS_STRIP_INSERTION_ERROR Strip insertion error
                RBLE_GLP_SENSORSTATUS_STRIP_TYPE_INCORRECT  Strip type incorrect for device
                RBLE_GLP_SENSORSTATUS_RESULT_HIGH              Sensor result higher than the device can
                                                           process
                RBLE_GLP_SENSORSTATUS_RESULT_LOW               Sensor result lower than the device can
                                                           process
                RBLE_GLP_SENSORSTATUS_TEMPERATURE_HIGH         Sensor temperature too high for valid
                                                           test/result at time of measurement
                RBLE_GLP_SENSORSTATUS_TEMPERATURE_LOW          Sensor temperature too low for valid
                                                           test/result at time of measurement
                RBLE_GLP_SENSORSTATUS_READ_INTERRUPTED         Sensor read interrupted because strip was
                                                           pulled too soon at time of measurement
                RBLE_GLP_SENSORSTATUS_GENERAL_DEVICE_FAULT  General device fault has occurred in the
                                                           sensor
                RBLE_GLP_SENSORSTATUS_TIME_FAULT               Time fault has occurred in the sensor and
time
                                                           may be inaccurate
                * Select multiple cases by using OR for example,
                *                                    (RBLE_GLP_SENSORSTATUS_DEVICE_BATTERY_LOW       |
        RBLE_GLP_SENSORSTATUS_TEMPERATURE_LOW)
                * One must be sure that the Glucose feature value indicates support for the above
                * conditions for the GATT Client to report the matching status settings.
                * The Glucose feature, unfortunately, requires coding and building the
                * RL78 HEX file. The current RL78 project for the Glucose meter supports all the above
        */

        sMderFloat glucConcentration; /* Glucose concentration as an as an MDER_SFLOAT */
        RBLE_DATE_TIME timestamp;
        time_t unixTimeStamp;           /* We will use this to help correct time stamps on a set time
*/
        sMderFloat carbohydateAmount; /* Units are grams as an MDER_SFLOAT */
        uint8_t carbohydrateId;                 /* One and only one of
                                                RBLE_GLP_CARBOHYDRATEID_BREAKFAST
                                                RBLE_GLP_CARBOHYDRATEID_LUNCH,
                                                RBLE_GLP_CARBOHYDRATEID_DINNER,
                                                RBLE_GLP_CARBOHYDRATEID_SNACK,
                                                RBLE_GLP_CARBOHYDRATEID_DRINK,
                                                RBLE_GLP_CARBOHYDRATEID_SUPPER,
                                                RBLE_GLP_CARBOHYDRATEID_BRUNCH */
        uint8_t meal;                 /* One and only one of
                                                RBLE_GLP_MEAL_PREPRANDIAL,
                                                RBLE_GLP_MEAL_POSTPRANDIAL,
                                                RBLE_GLP_MEAL_FASTING,
                                                RBLE_GLP_MEAL_CASUAL,
                                                RBLE_GLP_MEAL_BEDTIME */
        uint8_t tester;                       /* One and only one of
                                                RBLE_GLP_TESTER_SELF,
                                                RBLE_GLP_TESTER_HEALTH_CARE_PRO,
                                                RBLE_GLP_TESTER_LAB_TEST,
                                                RBLE_GLP_TESTER_NOT_AVAILABLE */
        uint8_t health;                       /* One and only one of
                                                RBLE_GLP_HEALTH_MINOR_ISSUE
                                                RBLE_GLP_HEALTH_MAJOR_ISSUE,
                                                RBLE_GLP_HEALTH_DURING_MENSES,
                                                RBLE_GLP_HEALTH_UNDER_STRESS,
                                                RBLE_GLP_HEALTH_NO_ISSUE,
                                                RBLE_GLP_HEALTH_NOT_AVAILABLE */
        uint8_t medicationId;         /* One and only one of
                                                RBLE_GLP_MEDICATIONID_RAPID_INSULIN,
                                                RBLE_GLP_MEDICATIONID_SHORT_INSULIN,
                                                RBLE_GLP_MEDICATIONID_INTERMEDIATE_INSULIN,
                                                RBLE_GLP_MEDICATIONID_LONG_INSULIN,
                                                RBLE_GLP_MEDICATIONID_PREMIXED_INSULIN */
        uint16_t excerciseDuration;   /* In seconds */
        uint8_t exerciseIntensity;    /* Percent of maximum */
        sMderFloat medicationAmount;  /* Units are in ml or mg as an MDER_SFLOAT */
        sMderFloat hbA1C;             /* Units are percent as an MDER_SFLOAT */
} sGlMeasurement;
```

The Glucose Service defines most of the units in kg and liters. The values entered in this structure are properly scaled to the units demanded by the profile.

8.4 Shutdown

The demo application currently waits `SECONDS_TO_DISCONNECT` seconds after the last detectable peer GATT action before sending a disconnect. A further detectable GATT action resets the time to disconnect to `SECONDS_TO_DISCONNECT` seconds. However, the RL78/G1D MCU does not detect peer READs or service discovery attempts. The application is free to change this `SECONDS_TO_DISCONNECT` default value and behavior to disconnect after the last measurement.

Once the disconnect is sent, the demo application waits `SECONDS_TO_REBROADCAST` seconds before advertising again. A production application will probably not want to do that but invoke a shutdown instead. Both `SECONDS_TO_DISCONNECT` and `SECONDS_TO_REBROADCAST` are defined in `*Profile_App.h` and are configurable. The seconds to disconnect should not be less than 5.

In the demo application pressing the S4 switch manually shuts down the application by calling the `btleAdapterShutDown()` method. This method saves any needed data to flash and terminates the **embedded loop**.

9. Testing the Application

Once the Demo Project has been built and loaded onto the Synergy MCU and the supporting `RL78 HEX` file onto the RL78/G1D MCU, applying power will start the application. Any Continua compliant PHG application supporting the given profile will be able to receive the measurements. Lamprey Networks provides DeviceFhir and Health@Home PHG on Android and iOS (see References). These applications are free and are available in the respective stores.

If using a Continua compliant PHG, one will need to discover the device and connect/pair with it. The demos start advertising shortly after power is applied and continue to advertise until shutdown or a PHG connects to it.

10. Continua Compliance Testing

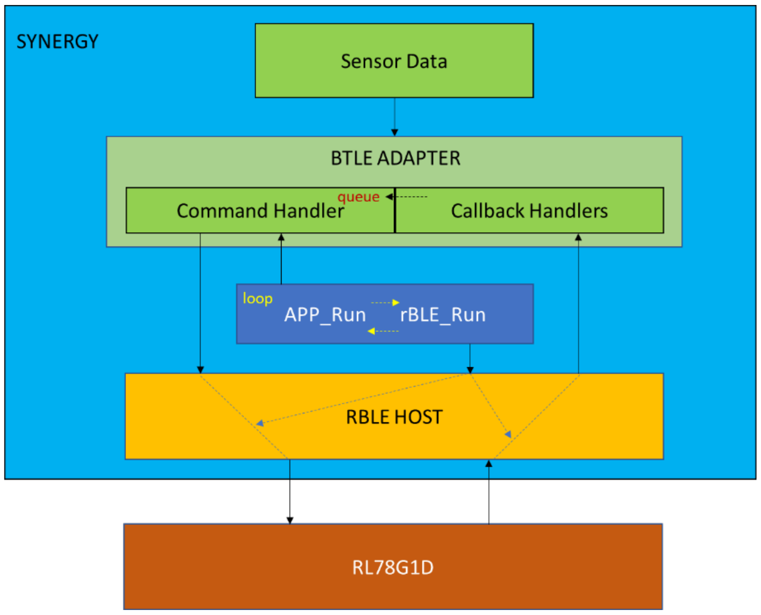
Compliance testing is done by running the Continua Test suit. It is available for free from the PCHA web site. The compliance tests consist of configuring the test parameters and running the set of tests appropriate to the device type being tested. Compliance testing also requires interoperability testing with Continua Compliant Gateways.

The test suite runs on a Windows PC. It requires an external Bluetooth Low Energy dongle. The test suite is harder to use than the PHG applications noted previously as it is designed to test various features of the profiles and thus requires considerable Bluetooth Low Energy and profile-specific knowledge.

11. As Built Architecture (Applies to all Continua Agents)

The basic architecture of the BTLE Adapter is the same for all supported profiles and is quite simple. The BTLE Adapter provides the profile and Continua intelligence of the Bluetooth functionality. It communicates with the RL78/G1D MCU through the RBLE HOST. The communication between the BTLE Adapter and the RBLE HOST is done through a rapidly repeating sequence of two calls; one done into the BTLE Adapter command handler followed by a call into the RBLE HOST. This repeating sequence links the BTLE Adapter to the RBLE HOST.

The design of the BTLE Adapter makes it easy to port between any platform that has an RBLE HOST as most of its code and calls are standard C. The only platform-dependent methods in the BTLE Adapter are calls to the Real Time Clock to get the current time and calls to save data to flash.



`rBLE_Run()` handles both incoming activity from the RL78/G1D MCU and activity going to the RL78/G1D MCU, thus the need to repeatedly call the method. `APP_Run()` calls the command handler in the BTLE Adapter and if a command exists in the queue, it is handled. Almost all the time **handling** requires making an `RBLE_*` call into the RL78/G1D MCU. After the call, the command handler returns and `rBLE_Run()` allows the RBLE HOST to do its work. RL78/G1D MCU responses are sent by the `rBLE_Run()` to the BTLE Adapter's registered callback handlers. There are callback handlers for each of the basic BTLE engines, GAP, SM (Security Manager), and GATT/Profile. Handling the callback usually results in placing a command into the BTLE Adapter Command Handler buffer and returning, allowing the callback to complete.

The Synergy Application need only start the BTLE Adapter, start the main loop, and provide the BTLE Adapter with sensor data. A shut down method is called when the application is going to power off. The shutdown disconnects any connected peer, saves data to flash, and terminates the loop. It is assumed that this method is only called when the application intends to power down. The BTLE Adapter can only be restarted by a hardware reset or power up.

12. Inner Workings of the BTLE Adapter

An application may require modifications of the BTLE Adapter. The following gives a description of how the Adapter works which is really an exercise in understanding how the RL78/G1D API works. Only one of the RL78/G1D methods is NOT asynchronous; thus, one typically makes a call and waits for a response in a callback. Since a timely return is often needed in callbacks to assure the receive thread is not blocked, the next call is not made at that time but a command to perform the next call is placed in a queue allowing the callback to return. A command processor reads the next command and acts accordingly, usually resulting in another call into the RL78/G1D MCU.

The BTLE Adapter consists of the following modules:

- **`APP_Run():`** Command Handler, reads the next command in the queue and performs the action indicated by the command, typically an RBLE call into the RL78/G1D MCU that invokes either a GAP, Security Manager (SM), or GATT/Profile callback. File: `*ProfileCommandHandler.c`
- **GAP Callback Handler**: asynchronous response to RBLE GAP calls. BTLE Adapter queues a command to handle the response or perform the next action. File: `*ProfileGAPCallbackHandlers.c`
- **GATT/Profile Callback Handler**: asynchronous response to RBLE GATT or Profile calls. BTLE Adapter queues a

command to handle the response or perform the next action. File: `*Profile*CallbackHandlers.c`

- **SM Callback Handler**: asynchronous response to RBLE SM calls. BTLE Adapter queues a command to handle the response or perform the next action. File: `*ProfileSMCallbackHandlers.c`

Some callbacks are signaled unsolicited, for example when the client writes a characteristic or invokes an RACP transaction or begins pairing.

13. Infinite Embedded Loop

The `rBLE HOST` is polling and must be continuously called by the application which, in this case, is the BTLE Adapter. The call allows the rBLE HOST to handle messages coming from the RL78/G1D MCU and dispatch them to the BTLE Adapter as well handling messages from the BTLE Adapter being delivered to the RL78/GID MCU. The `rBLE HOST` method `rBLE_Run()` handles these operations.

The polling nature of `rBLE_Run()` means that as much as possible one wants to perform a single operation and then call `rBLE_Run()`. Thus, at the core of the BTLE Adapter is an infinite while loop consisting of two methods and a brief wait. An initialization call triggers the process. The sequence appears as follows:

```
APP_Init()
while (1)
{
  if(APP_Run() == FALSE)
  {
        rBLE_Run();
        tx_thread_sleep(1);
  }
  else
  {
        break;
  }
}
```

`APP_Init()` initiates the process by calling `RBLE_Init()` which takes as a parameter a BTLE Adapter-provided callback function. The callback is signaled when the RL78/G1D MCU is active. The response is asynchronous, and the call returns immediately and the **infinite embedded loop** begins. It is important to understand that both the call to the RL78/G1D MCU and the asynchronous response from the RL78/G1D MCU will not proceed until `rBLE_Run()` is called at least once. When the BTLE Adapter is signaled that the `APP_Init()` has successfully completed, the BTLE Adapter is ready for the next stage which typically consists of making the next call to the RL78/G1D MCU. However, the BTLE Adapter does not make the call immediately but places a command in a queue for subsequent handling and returns. The queued commands are then processed when calling `APP_Run()`. Processing a command involves invoking an `RBLE_*` call, whose response is asynchronous, and returning. If the command queue is empty, the call returns immediately maintaining the calls into `rBLE_Run()`.

Unsolicited asynchronous events are also triggered by peer actions, such as a pairing request. Often these actions require that an appropriate response is generated. Again, the BTLE Adapter does this process by invoking a command for the response and returning, and handling the response when `APP_Run()` gets that command.

14. Bluetooth Initialization

Before the RL78/G1D MCU is ready to communicate with the peer, aspects of the Bluetooth behavior need to be configured such as the security model and use of privacy. This initialization is done before the device starts advertising. By exposing advertisements, the device is stating that it is ready to handle actions from the peer.

15. The BTLE Adapter Command-Callback Sequence

The following sequence shows the initialization of the RL78/G1D MCU up to the point where is begins advertising. All these transactions are just between the Synergy and the RL78/G1D MCU; no peer is involved.

15.1 Start Up

The BTLE Adapter begins by setting up the RL78/G1D MCU to handle communications in the desired manner. The sequence of methods, commands, subsequent events, callbacks, and commands is as follows:

```
APP_Init()
    RBLE_Init(APP_RBLE_Callback) triggers RBLE_MODE_ACTIVE
      APP_RBLE_Callback(mode)
```

```
          RBLE_MODE_ACTIVE: queue: GAP_RESET_CMD
APP_Run()
    GAP_RESET_CMD
      RBLE_GAP_Reset(APP_GAP_Callback,         APP_SM_Callback)         triggers
RBLE_GAP_EVENT_RESET_RESULT
          APP_GAP_Callback(event)
            RBLE_GAP_EVENT_RESET_RESULT: queue: GAP_GET_DEVICE_INFO_CMD
    GAP_GET_DEVICE_INFO_CMD
      RBLE_GAP_Get_Device_Info() triggers RBLE_GAP_EVENT_GET_DEVICE_INFO_COMP
          APP_GAP_Callback(event)
            RBLE_GAP_EVENT_GET_DEVICE_INFO_COMP: queue: GAP_SET_BONDING_MODE_CMD
    GAP_SET_BONDING_MODE_CMD
      RBLE_GAP_Set_Bonding_Mode() triggers RBLE_GAP_EVENT_SET_BONDING_MODE_COMP
          APP_GAP_Callback(event)
            RBLE_GAP_EVENT_GET_DEVICE_INFO_COMP:                           queue:
GAP_SET_SECURITY_REQUEST_CMD
    GAP_SET_SECURITY_REQUEST_CMD
      RBLE_GAP_Set_Security_Request()                                    triggers
RBLE_GAP_EVENT_SET_SECURITY_REQUEST_COMP
          APP_GAP_Callback(event)
            RBLE_GAP_EVENT_SET_SECURITY_REQUEST_COMP: queue: SM_SET_KEY_CMD
    SM_SET_KEY_CMD
      RBLE_SM_Set_Key() triggers RBLE_SM_EVENT_SET_CNF
          APP_SM_Callback(event)
            RBLE_SM_EVENT_SET_CNF: queue: GAP_BROADCAST_ENABLE_CMD
    GAP_BROADCAST_ENABLE_CMD
      The contents of the advertisement are configured here.
      RBLE_GAP_Broadcast_Enable()  --- Triggers no event; advertising starts
```

At this point nothing further will happen until a peer collector detects the advertisements and invokes a connection event.

15.2 Connection

A connection will trigger an `RBLE_GAP_EVENT_CONNECTION_COMP` event.

```
          APP_GAP_Callback(event)
            RBLE_GAP_EVENT_CONNECTION_COMP
            Update the current time characteristic in the CTS on the RL78
            RBLE_GAP_Set_Data() triggers RBLE_GATT_EVENT_SET_DATA_COMP
            This method sets the local value of the CTS on the RL78. Does not
involve the peer.

            Checks if the connection event is from a known peer collector
            Yes: expect an attempt to begin encryption
            No: Delete the current peer device information (an application may
      want to change this
                Behavior such that it will only work with a new device on user
      consent).
            Option: Request the peer start encryption if paired, otherwise start
      pairing if not
                                                   queue: SM_START_ENC_CMD
            Enable the desired profile               queue:
      (profile)_SENSOR_ENABLE_CMD
```

A Continua compliant collector, if not paired will do service discovery and read the Device Information Service characteristics and the Current Time. The BTLE Adapter is not aware of any of these processes as the RL78/G1D MCU stack does not signal such events to the upper layers. If pairing takes place the BTLE Adapter will be signaled several times. The BTLE Adapter will obtain at least long-term keys and IRKs for privacy address resolution as well as deliver corresponding keys to the peer. Since the BTLE Adapter is always playing the role of slave, it will be the BTLE Adapter long term keys used for pairing/encryption.

After pairing is done, the BTLE Adapter will obtain events if the peer attempts to set the time and when it enables indications and/or notifications. When enabling and pairing are complete, the BTLE Adapter indicates that it is ready by setting the `sDeviceInfo.isReady` field to TRUE which tells the measurement handler measurement data can now be transmitted to the peer. There will also be profile-dependent `sDeviceInfo.is*Enabled` fields set to TRUE, and these must be set to TRUE before `sDeviceInfo.isReady` can be set to TRUE. In the Glucose profile only RACP transactions will start the transmission of measurements.

If the profile supports Record Access Control Point transactions, the BTLE Adapter will receive events for that as well.

If already paired, the BTLE Adapter will receive an `RBLE_ENC_START_IND` event. If the BTLE Adapter has been previously enabled, the `sDeviceInfo.isReady` field is set to TRUE allowing measurements to be transmitted to the peer.

15.3 Application Modifications

Some of the behaviors an application may want to modify are as follows:

15.3.1 Connection

The default application will connect to any new device even if it is paired to another device. If the connection attempt is from a device it does not know, and it has saved information from another device, it will delete that saved information and proceed with the connection. An application may want to prevent the connection if already paired with another device. To connect with a new device, the application may require you to delete the current pairing.

15.3.2 Disconnect and Shutdown

On a disconnect, the default application will not shutdown but remain idle for a period of 20 seconds (configurable) before it starts advertising again unless the user shuts it down. An application may want to shutdown automatically on disconnect to save power.

15.3.3 Starting the BTLE Functionality

The default application starts the Bluetooth functionality on power up. An application may want to wait until a measurement is taken and passed to the BTLE Adapter before starting the Bluetooth functionality. The application will need to invoke the `APP_Init()` and subsequent **Infinite Embedded Loop** at the appropriate time.

16. RL78/G1D Programming

The application will need to set values on the RL78/G1D MCU. These values are the application-specific Device Information Service fields, the profile feature characteristics, and the Bluetooth Address. What the application needs to configure is described in the following sections.

16.1 Profile Selection

First, you configure the correct profiles. This work is done in the file `prf_sel.h`. The selections are done in the following section of code:

```
#ifndef USE_SAMPLE_PROFILE
#define PRF_SEL_PXPM    0   /* Proximity Profile Monitor role */
#define PRF_SEL_PXPR    0   /* Proximity Profile Reporter role */
#define PRF_SEL_FMPL    0   /* Find Me Profile Locator role */
#define PRF_SEL_FMPT    0   /* Find Me Profile Target role */
#define PRF_SEL_HTPC    0   /* Health Thermometer Profile Collector role */
#define PRF_SEL_HTPT    0   /* Health Thermometer Profile Thermometer role */
#define PRF_SEL_BLPC    0   /* Blood Pressure Profile Collector role */
#define PRF_SEL_BLPS    0   /* Blood Pressure Profile Sensor role */
#define PRF_SEL_HGHD    0   /* HID over GATT Profile HID Device role */
#define PRF_SEL_HGBH    0   /* HID over GATT Profile Boot Host role */
#define PRF_SEL_HGRH    0   /* HID over GATT Profile Report Host role */
#define PRF_SEL_SPPC    0   /* Scan Parameters Profile Scan Client role */
#define PRF_SEL_SPPS    0   /* Scan Parameters Profile Scan Server role */
#define PRF_SEL_HRPC    0   /* Heart Rate Profile Collector role */
#define PRF_SEL_HRPS    0   /* Heart Rate Profile Sensor role */
#define PRF_SEL_CSCC     0   /* Cycling Speed and Cadence Profile Collector role
*/
#define PRF_SEL_CSCS    0   /* Cycling Speed and Cadence Profile Sensor role */
#define PRF_SEL_GLPC    0   /* Glucose Profile Collector role */
#define PRF_SEL_GLPS    1   /* Glucose Profile Sensor role */
#define PRF_SEL_CPPC    0   /* Cycling Power Profile Collector role */
#define PRF_SEL_CPPS    0   /* Cycling Power Profile Sensor role */
```

```
#define PRF_SEL_TIPC    0   /* Time Profile Client role */
#define PRF_SEL_TIPS    0   /* Time Profile Server role */
#define PRF_SEL_CTS     1   /* Current Time Server role */
#define PRF_SEL_ANPC    0   /* Alert Notification Profile Client role */
#define PRF_SEL_ANPS    0   /* Alert Notification Profile Server role */
#define PRF_SEL_LNPS    0   /* Location and Navigation Profile Sensor role */
#define PRF_SEL_LNPC    0   /* Location and Navigation Profile Collector role */
#define PRF_SEL_PASC    0   /* Phone Alert Status Profile Client role */
#define PRF_SEL_PASS    0   /* Phone Alert Status Profile Server role */
#define PRF_SEL_RSCC    0   /* Running Speed and Cadence Profile Collector role
*/
#define PRF_SEL_RSCS    0   /* Running Speed and Cadence Profile Sensor role */
#define PRF_SEL_PLXC    0   /* Pulse Oximeter Profile Collector role */
#define PRF_SEL_PLXS    0   /* Pulse Oximeter Profile Sensor role */
```

The highlighted lines are the only selections the application shall modify. Only one of these three can be set given the current design of the BTLE Adapter. The example above shows the setting for the Glucose Profile. The CTS must always be set. Note that the settings are done in the section `#ifndef USE_SAMPLE_PROFILE`. There are identical settings in the **#else** block which should be indicated as not being compiled. If not, the e^2 studio is not properly configured. It must be configured to **NOT** use the sample profile.

16.2 Device Information Service Settings

Once the `prf_sel.h` file is properly set, setting the Device Information fields is done in the file `prf_config.c`. This work is done in the section of code shown below. Be sure to modify the correct section related to the Device Information Service. There are pre-defined sections for each of the supported profiles. The application developer is free to delete any of the code **in the shown section** that is not in your profile.

```
#if (USE_DEV_INFO_SERVICE || USE_DEV_INFO_SERVICE_PNP)
/**********************************
 * Device Information service    *
 **********************************/
/** Device Information Service */
static const uint16_t dis_svc = RBLE_SVC_DEVICE_INFORMATION;

#if (USE_DEV_INFO_SERVICE)
/** System ID Characteristic */
static       const       struct       atts_char_desc       dis_sys_id_char       =
ATTS_CHAR(RBLE_GATT_CHAR_PROP_RD,
                                                                 DIS_HDL_SYS_ID_VAL,
                                                                 RBLE_CHAR_SYSTEM_ID);
/** System ID value (8octets) */
/** DIS CONFIGURE! User needs to enter it's company IEEE System Id here
  * Note that the entry is in little-endian. Least significant bytes first! */
#if(PRF_SEL_BLPS)
static const uint8_t dis_sys_id_desc[8] = { 0, 0, 0, 0xF0, 2, 3, 4, 1 };
#endif
#if(PRF_SEL_GLPS)
static const uint8_t dis_sys_id_desc[8] = { 0, 0, 0, 0xF4, 2, 3, 4, 1 };
#endif
#if(PRF_SEL_PLXS)
static const uint8_t dis_sys_id_desc[8] = { 0, 0, 0, 0xF2, 2, 3, 4, 1 };
#endif
#if(PRF_SEL_HTPT)
static const uint8_t dis_sys_id_desc[8] = { 0, 0, 0, 0xF3, 2, 3, 4, 1 };
#endif

/** Model Number Characteristic */
static       const       struct       atts_char_desc       dis_model_nb_char       =
ATTS_CHAR(RBLE_GATT_CHAR_PROP_RD,
                                                                  DIS_HDL_MODEL_NB_VAL,

RBLE_CHAR_MODEL_NUMBER_STRING);
```

```
/** Model Number String */
static const uint8_t dis_model_nb_desc[]    = "Synergy-12345-Demo";

/** Serial Number characteristic */
static      const      struct      atts_char_desc      dis_serial_nb_char      =
ATTS_CHAR(RBLE_GATT_CHAR_PROP_RD,
                                                                  DIS_HDL_SERIAL_NB_VAL,

RBLE_CHAR_SERIAL_NUMBER_STRING);
/** Serial Number String */
#if(PRF_SEL_BLPS)
static const uint8_t dis_serial_nb_desc[] = "13456-BPM-BTLE";
#endif
#if(PRF_SEL_GLPS)
static const uint8_t dis_serial_nb_desc[] = "13456-GLU-BTLE";
#endif
#if(PRF_SEL_PLXS)
static const uint8_t dis_serial_nb_desc[] = "13456-PLX-BTLE";
#endif
#if(PRF_SEL_HTPT)
static const uint8_t dis_serial_nb_desc[] = "13456-HTP-BTLE";
#endif
/** Firmware Revision Characteristic */
static      const      struct      atts_char_desc      dis_fw_rev_char      =
ATTS_CHAR(RBLE_GATT_CHAR_PROP_RD,
                                                                DIS_HDL_FW_REV_VAL,

RBLE_CHAR_FW_REVISION_STRING);
/** Firmware Revision String */
static const uint8_t dis_fw_rev_desc[] = "1.0.0";

/** Hardware Revision Characteristic */
static      const      struct      atts_char_desc      dis_hw_rev_char      =
ATTS_CHAR(RBLE_GATT_CHAR_PROP_RD,
                                                                DIS_HDL_HW_REV_VAL,

RBLE_CHAR_HW_REVISION_STRING);
/** Hardware Revision String **/
static const uint8_t dis_hw_rev_desc[] = "1.0.0";

/** Software Revision Characteristic */
static      const      struct      atts_char_desc      dis_sw_rev_char      =
ATTS_CHAR(RBLE_GATT_CHAR_PROP_RD,
                                                                DIS_HDL_SW_REV_VAL,

RBLE_CHAR_SW_REVISION_STRING);
/** Software Revision String */
static const uint8_t dis_sw_rev_desc[] = "1.0.0";

/** Manufacturer Name Characteristic **/
static      const      struct      atts_char_desc      dis_manuf_name_char      =
ATTS_CHAR(RBLE_GATT_CHAR_PROP_RD,
                                                              DIS_HDL_MANUF_NAME_VAL,

RBLE_CHAR_MANUF_NAME_STRING);
/** Manufacturer Name String */
static const uint8_t dis_manuf_name_desc[] = "Renesas Electronics";

/** IEEE 11073-20601 Regulatory Certification Data List */
static      const      struct      atts_char_desc      dis_ieee_certif_char      =
ATTS_CHAR(RBLE_GATT_CHAR_PROP_RD,
DIS_HDL_IEEE_CERTIF_VAL,

RBLE_CHAR_IEEE_CERTIF);
/** IEEE 11073-20601 Regulatory Certification Data List Value (Min:6octets) */
/** IEEE 11073-20601 Regulatory Certification Data List Value (Min:6octets) */
/** DIS CONFIGURE! User needs to enter Reg-Cert-Data-List string here. This is
not easy!
 * This is an ASN1 data structure which depends upon the specialization, continua
version, etc.
 * It is not in little endian format but 20601 Mder. It is as follows:

 *                      0x00, 0x02,   // Number of entries (For Continua always 2)
 *                        0x00, 0x12,   // Length (will increase if more than one
specialization)
 *                                   // note that the length fields do NOT include
the length field
 *                                    // or anything before it.
                        0x02,         // organization defining reg cert data entry
(2 = Continua)
                        0x01,           // Continua struct type 1 = certification
entries
                        0x00, 0x08,   // length of struct type 1 (will increase if
more than one specialization)
                        0x06, 0x01,   // Continua version (currently 6.1)
                        0x00, 0x01,   // number of certified device specializations
                        0x00, 0x02,   // length of list
                        0x80, 0x07,   // device type. Tcode (0x80 = BTLE)
                                      // value of MDC_DEV_SPEC_PROFILE_* - 4096;
                                       *  weight scale = 0x0F
                                       *  (BTLE advanced weigh scale is 20601 bca =
0x14
                                       *  pulse oximeter =0x04
                                       *  blood pressure = 0x07
                                       *  thermometer = 0x08
                                       *  glucose = 0x11
                        0x02,         // Continua reg cert entry
                        0x02,         // Continua struct type 2 = regulation
                        0x00, 0x02,   // length
                        0x80,  0x00};  // unregulated (00 00 = regulated, 80 00 =
unregulated)
                                      // ASN1 Mder BITs field
  The application can leave this alone; it is properly configured for each profile.
Except one might want to change the regulation status */
#if(PRF_SEL_BLPS)
    static const uint8_t dis_ieee_certif_desc[] = { 0, 2, 0, 0x12, 2, 1, 0, 8, 6,
1, 0, 1, 0, 2, 0x80, 7, 2, 2, 0, 2, 0x80, 0 };
#endif
#if(PRF_SEL_GLPS)
    static const uint8_t dis_ieee_certif_desc[] = { 0, 2, 0, 0x12, 2, 1, 0, 8, 6,
1, 0, 1, 0, 2, 0x80, 0x11, 2, 2, 0, 2, 0x80, 0 };
#endif
#if(PRF_SEL_PLXS)
    static const uint8_t dis_ieee_certif_desc[] = { 0, 2, 0, 0x12, 2, 1, 0, 8, 6,
1, 0, 1, 0, 2, 0x80, 4, 2, 2, 0, 2, 0x80, 0 };
#endif
#if(PRF_SEL_HTPT)
    static const uint8_t dis_ieee_certif_desc[] = { 0, 2, 0, 0x12, 2, 1, 0, 8, 6,
1, 0, 1, 0, 2, 0x80, 8, 2, 2, 0, 2, 0x80, 0 };
#endif
#endif /* #if (USE_DEV_INFO_SERVICE) */
```

16.3 Securing the Measurement Descriptors

If you want to be sure that measurements are not delivered unless pairing and encryption is in place, setting the security property of the measurement characteristic value descriptors to require encryption will assure attempts to write the descriptors return an **insufficient authentication** error unless pairing has taken place.

Below is an example for the Blood Pressure Measurement Characteristic configuration descriptor, highlighted in yellow:

```
/*  Blood Pressure Measurement characteristic definition*/
{ RBLE_DECL_CHARACTERISTIC,
    sizeof(blps_bldprs_meas_char),
    sizeof(blps_bldprs_meas_char),
    TASK_ATTID(TASK_BLPS, BLS_IDX_BLDPRS_MEAS_CHAR),
  RBLE_GATT_PERM_RD,
    (void *)&blps_bldprs_meas_char },
/* Blood Pressure Measurement Value */
{ RBLE_CHAR_BLOOD_PRESSURE_MEAS,
    sizeof(blps_packed_measurements),
    sizeof(blps_packed_measurements),
    TASK_ATTID(TASK_BLPS, BLS_IDX_BLDPRS_MEAS_VAL),
    RBLE_GATT_PERM_NI,
    (void *)&blps_packed_measurements[0] },
/* Blood Pressure Measurement Indication Cfg Value */
{ RBLE_DESC_CLIENT_CHAR_CONF,
    sizeof(blps_bldprs_meas_ind_en),
    sizeof(blps_bldprs_meas_ind_en),
    TASK_ATTID(TASK_BLPS, BLS_IDX_BLDPRS_MEAS_IND_CFG),
   (RBLE_GATT_PERM_RD|RBLE_GATT_PERM_WR_UNAUTH|RBLE_GATT_PERM_ENC),
   (void *)&blps_bldprs_meas_ind_en },
```

This permission has already been added to all the relevant profile characteristics that need security doe Continua compliance. In the pulse oximeter and glucose profiles, the RACP characteristic configuration descriptors, in addition to the measurement characteristic configuration descriptors also need to be secured.

16.4 Hiding unsupported Characteristics

One can add the field `RBLE_GATT_PERM_HIDE` to characteristics or services that you don't want to support. For example, the optional battery service has been hidden which is included by default when one selects the pulse oximeter profile. It is shown below:

```
{ RBLE_DECL_PRIMARY_SERVICE,
        sizeof(bas_svc1),
        sizeof(bas_svc1),
        TASK_ATTID(TASK_SVC, BAS1_IDX_SVC),
        RBLE_GATT_PERM_HIDE|RBLE_GATT_PERM_RD,
        (void *)&bas_svc1 },
```

The intermediate measurement characteristic of the Blood Pressure Service is not supported in the BTLE Adapter. This characteristic is not hidden but an application should probably do so unless one wants to add support for the intermediary measurement.

16.5 Bluetooth Address

The Bluetooth address is set in the file `config.h`. They are defined here:

```
/* Public Device Address for test and development use */
#if(PRF_SEL_BLPS)
    #define CFG_TEST_BDADDR     {{0xbc, 0x9a, 0x78, 0x56, 0x34, 0x12}}
#endif
#if(PRF_SEL_GLPS)
    #define CFG_TEST_BDADDR     {{0xbf, 0x9a, 0x78, 0x56, 0x34, 0x12}}
#endif
#if(PRF_SEL_PLXS)
    #define CFG_TEST_BDADDR     {{0xbd, 0x9a, 0x78, 0x56, 0x34, 0x12}}
#endif
#if(PRF_SEL_HTPT)
```

```
    #define CFG_TEST_BDADDR     {{0xbe, 0x9a, 0x78, 0x56, 0x34, 0x12}}
#endif
```

The demo has set an address for each of the profiles. You must set an address appropriate for your application. You can delete the code that is not relevant for the profile of interest.

17. Activity Monitor Service

17.1 Profile

The Activity Monitor (AM) peripheral will support the AM Service and the Device Information Service. The Device Information Service will contain at least the system Id, Manufacturer Name, Model number, Serial number, and for Continua the Regulation certification fields. The AM peripheral will include the AM Service UUID in its general and limited discovery advertisements. On connection or re-connection, the AM peripheral will not send any data until its Sensor Current Time Characteristic has been read and the toggle characteristic has been set. The collector will read the Sensor Current Time Characteristic before writing to the toggle characteristic. When the AM peripheral has transferred all its data and is not actively sending real time data it will disconnect from the collector.

Security and pairing requirements are recommended but optional for the AM peripheral. It is required for PCHA peripherals.

The AM Collector will be able to discover all the required services, characteristics, and characteristic descriptors in the AM peripheral. The AM Collector will be able to discover all the optional characteristics and descriptors in the AM Service. The AM Collector will always read the Sensor Current Time Characteristic of AM peripheral when first connecting or reconnecting (security may be required to be in place). The AM Collector shall be able to set the sensor current time if requested by the AM peripheral. The AM Collector will support the relative, absolute, and Base Offset times.

17.2 Toggle

The AM Service has a toggle characteristic (may be replaced by some form of an RACP like the Pulse Oximeter) that allows the collector to indicate to the peripheral it is ready to receive data. The Collector writes to this toggle characteristic after it enables the characteristics it wants to receive.

17.3 Sessions

The AM reports activities such as walk, run, swim, sleep, rest, multiple, etc. in sessions. Each session scopes an activity. The AM reports a session and data associated with that session.

Typically, a session is a time period where a certain activity is recorded. What determines a session start and end is up to the application. It may be determined interactively by a user, for example the start and end of a training run or mountain climb or automatically by the device. Some devices may be able to automatically sense activities and invoke sessions in that manner while other devices will automatically start and end sessions by the clock, for example on the hour, day, or week, regardless of the current activity. In the latter case a single run from the perspective of the user may cross two sessions. The athlete who is focused on the results of training would likely obtain an AM that allows one to scope the training whereas the person who is interested in increasing overall fitness might be more interested in an AM that runs continuously and gives daily or hourly reports of the activity and calories burned.

17.4 Data

In this profile the types of data reported by an AM is either some type of count or rate. Count data typically grows with time such as steps taken, calories burned, distance walked, and so on. Rate data typically has well defined instantaneous values such as the cadence, speed, heart or breathing rate, and so on, and is suited for statistics (averages, maxima, minima). The AM service reports these two different data types in their own characteristics.

17.5 Session Reporting Methods

An AM can report data to the collector in two different ways; store and forward (summary) and continuous (instantaneous) monitoring.

17.5.1 Store and Forward or Summary Mode

Store and forward means that the activity is performed and the results are reported as summaries at the end. The data would consist of statistics (means, maxima, and minima) and totals. Store and forward AMs typically accumulate the data offline and upload the results to the collector later. The upload would consist of the session describing the activity type and session length and the subsequent data. For a step counting AM that data might consist of the total number of steps, the total distance covered, the total calories burned, the total amount of time performing the activity, the average pace, the maximum pace, the average heart rate, and the maximum heart rate.

In the store and forward case, the session carries the time stamp of all the data associated with it. The time stamp of the session is defined as the time at the **start** of the session. The time stamps are only reported in the session characteristics and not in the summary data characteristics. A session has an identifier within a connection. Data belonging to that session contains that identifier. An example reporting sequence would appear as follows:

```
Session 1 (2015/12/24 06:00:00, walk, 30 minutes, identifier 1)
        data (identifier 1, steps 2050, activity time 25 minutes, calories 107, distance 1700 m)
        data (identifier 1, mean pulse 92, mean speed 1.1 m/s, mean intensity 65%)
        data (identifier 1, max pulse 121, max speed 2.2 m/s, max intensity 72%)
Session 2 (2015/12/24 06:30:00, run, 30 minutes, identifier 2)
        data (identifier 2, steps 4250, activity time 28 minutes, calories 337, distance 5100 m)
        data (identifier 2, mean pulse 134, mean speed 3.0 m/s, mean intensity 80%)
        data (identifier 2, max pulse 151, max speed 3.2 m/s, max intensity 88%)
Session 3 (2015/12/24 07:00:00, walk, 60 minutes, identifier 3)
        data (identifier 3, steps 163, activity time 2 minutes, calories 11, distance 110 m)
```

17.5.2 Continuous Monitoring Mode

Continuous monitoring means instantaneous and/or cumulative values are reporting during the session. It does not mean that the AM runs continuously (though it may). In the continuous monitoring case the session does not contain a duration and all data contains time stamps. At what time these instantaneous and/or cumulative values are reported is up to the application, though they often tend to be periodic. Sessions reported in this manner have a flag to indicate that it is in continuous monitoring mode. When a session completes, store and forward reporting may follow. Note that the store and forward data will be reported in its own session. The application may upload three weeks of continuous monitoring data and then report the three weeks as summary data. The monitor may also support uploading the data to the collector in real time.

The following example illustrates reporting cumulative or current values every minute:

```
Session continuous (2015/12/24 06:00:00, walk, identifier 1)
        data (id 1, 2015/12/24 06:01:00, steps x, activity time t min, calories c, distance x m)
        data (id 1, 2015/12/24 06:01:00 pulse PR, speed v m/s, intensity I%)
        data (id 1, 2015/12/24 06:02:00, steps x, activity time t min, calories c, distance x m)
        data (id 1, 2015/12/24 06:02:00 pulse PR, speed v m/s, intensity I%)
                … (more data)
Session continuous (2015/12/24 06:30:00, run, identifier 2)
        data (id 2, 2015/12/24 06:31:00, steps x, activity time t min, calories c, distance x m)
        data (id 2, 2015/12/24 06:31:00 pulse PR, speed v m/s, intensity I%)
        data (id 2, 2015/12/24 06:32:00, steps x, activity time t min, calories c, distance x m)
        data (id 2, 2015/12/24 06:32:00 pulse PR, speed v m/s, intensity I%)
                … (more data)
```

Followed by summary data:

```
Session summary (2015/12/24 06:00:00, walk, 30 minutes, identifier 3)
        data (identifier 3, steps 2050, activity time 25 minutes, calories 107, distance 1700 m)
        data (identifier 3, mean pulse 92, mean speed 1.1 m/s, mean intensity 65%)
        data (identifier 3, max pulse 121, max speed 2.2 m/s, max intensity 72%)
Session summary (2015/12/24 06:30:00, run, 30 minutes, identifier 4)
        data (identifier 4, steps 4250, activity time 28 minutes, calories 337, distance 5100 m)
        data (identifier 4, mean pulse 134, mean speed 3.0 m/s, mean intensity 80%)
        data (identifier 4, max pulse 151, max speed 3.2 m/s, max intensity 88%)
```

It is also allowable to deliver the data in the following manner:

```
Session continuous (2015/12/24 06:00:00, walk, identifier 1)
        data (id 1, 2015/12/24 06:01:00, steps x, activity time t min, calories c, distance x m)
        data (id 1, 2015/12/24 06:01:00 pulse PR, speed v m/s, intensity I%)
        data (id 1, 2015/12/24 06:02:00, steps x, activity time t min, calories c, distance x m)
        data (id 1, 2015/12/24 06:02:00 pulse PR, speed v m/s, intensity I%)
                … (more data)
Session summary (2015/12/24 06:00:00, walk, 30 minutes, identifier 2)
        data (identifier 2, steps 2050, activity time 25 minutes, calories 107, distance 1700 m)
        data (identifier 2, mean pulse 92, mean speed 1.1 m/s, mean intensity 65%)
        data (identifier 2, max pulse 121, max speed 2.2 m/s, max intensity 72%)
```

Session continuous (2015/12/24 06:30:00, run, identifier 3)
data (id 3, 2015/12/24 06:31:00, steps x, activity time t min, calories c, distance x m)
data (id 3, 2015/12/24 06:31:00 pulse PR, speed v m/s, intensity I%)
data (id 3, 2015/12/24 06:32:00, steps x, activity time t min, calories c, distance x m)
data (id 3, 2015/12/24 06:32:00 pulse PR, speed v m/s, intensity I%)
… (more data)
Session summary (2015/12/24 06:30:00, run, 30 minutes, identifier 4)
data (identifier 4, steps 4250, activity time 28 minutes, calories 337, distance 5100 m)
data (identifier 4, mean pulse 134, mean speed 3.0 m/s, mean intensity 80%)
data (identifier 4, max pulse 151, max speed 3.2 m/s, max intensity 88%)

17.6 Time stamps

This specification introduces the relative time stamp and a base offset time stamp following IEEE 11073 20601. It does not use just the absolute time stamp as in the BLE Date Time Characteristic. Which type of time stamp the device is using is determined by a flag in the required Sensor Current Time Characteristic of the Activity Monitor service.

17.6.1 Relative time

The use of the relative time stamp simplifies the time keeping code by the peripheral as the translation to wall clock time is done by the collector and there is no consideration for the time zone. The value is set to 0 or some other value at the factory (and perhaps after a battery change) and increments with time. The units are in seconds but the value is recorded in a 32 bit number so it will take 136 years for the counter to recycle if the device chooses a start value of 0. For the collector to map the relative time to wall clock time the Activity monitor service includes a Sensor Current Time Characteristic. The collector reads the current **tick** value and compares it with its current wall clock time. With this knowledge, the relative time stamps in any of the characteristic values can be converted to wall clock times. For example, if the collector reads the current time at Dec 24th 2015 at 11:00:00 wall clock time and reads a value of 120 in the Sensor Current time characteristic and then receives a data characteristic with a time stamp value of 60 it knows that the characteristic is reporting something that occurred 60 seconds earlier thus it was at wall clock time of Dec 24th 2015 at 10:59:00.

A collector will not set a peripheral’s relative time clock.

17.6.2 Base offset time

The Base offset time stamp uses a simplified form of the IEEE 11073 20601 Base Offset time. The IEEE Base Offset time stamp reports the time as a base time in UTC plus an offset to the local time. While the IEEE standard allows the base time to be something other than UTC, this standard requires the base time to always be UTC aligned. In addition, this specification will not include the 2 byte fractional component and the base time of 0 corresponds to January 1st 2000 at 00:00:00 Z instead of January 1st 1900 at 00:00:00 Z. The device may want for the collector to set this time. The device indicates to the collector that it wants its time set by setting a flag in the Sensor Current Time Characteristic. The AM collector is required to read this characteristic after connection and any required security is in place and set the time if asked. Any subsequent readings of the Sensor Current Time Characteristic shall show the flag cleared unless the device determines that it needs its time set once again.

17.6.3 Stored data and time changes

If the AM peripheral has stored data and still allows it’s time to be set, it shall adjust the time stamps of all the stored data such that the stored data is on the same timeline as the newly set value. That adjustment will require the AM peripheral recording the difference between the time prior to the setting and the new time. The AM peripheral shall make this adjustment regardless of the reason for the change in the timeline (setting by the collector or by the user). If the timeline of the stored data cannot be adjusted for some reason such as a for an unexpected power loss, the stored data shall be marked with a time fault flag indicating to the collector that the time line of the data cannot be recovered.

17.6.4 Time faults

Any device with a real time clock may suffer a time fault where the time line is lost. This loss may occur after an error or an unexpected power loss such as removal of the battery while running. Typically, a time fault is not a problem since the collector can always correct the time stamps. However, if there is stored data that was generated before the time fault, there is no way the collector can recover the time line of the stored data and it is impossible to covert or correct the values to wall clock time. In the case of Base Offset time, the collector still cannot recover the time line but the stored time stamps can be easily converted to a wall clock time. However, there is no way for the collector to validate the reliability of the time stamps. Data affected by a time fault are therefore indicated with a time fault flag.

17.7 Procedure

The AM peripheral shall send advertisements when it has data to upload. When this attempt is started it is up to the application. It may, for example, be on a clock or determined by the user. The UUID of the AM service shall be present in the advertisement data in the limited and general discovery modes. The friendly name should be present in the limited and general discovery modes. Otherwise this specification places no additional restrictions on the frequency and lengths of the advertising events beyond that specified by the core specification.

If it is a first time connection and the collector wants to connect to the AM, the collector will discover at least all the required services and the required and optional characteristics and descriptors of the required services. If the AM peripheral does not support pairing it will indicate so with the error code 0x18 ‘pairing not allowed’ when the collector attempts to pair but it shall not disconnect. If the AM peripheral sends error code 0x18 the collector may continue without pairing, otherwise the AM collector disconnects. After the pairing stage the collector shall read the Current Time Characteristic of the AM Service. If the AM peripheral requests it’s time to be set and the AM peripheral supports a Base Offset time, the collector should set the time within 10 seconds. (What to do if it doesn’t?)

The collector shall then write to the AM service’s characteristic descriptors to enable indications and notifications. The AM peripheral that pairs will save the enabled state for that collector.

On a reconnection the collector will read the Current Time Characteristic as soon as any security measures have been completed. The AM peripheral will not send any data until the Current Time Characteristic has been read.

The conditions under which the data is deleted from the AM peripheral is up to the application.

17.8 Summary

Activity Monitor Service Primary Service (128 bit UUID) read permission, no security CurrentTimes Characteristic (128 bit UUID) read and write permission, unauthenticated pairing.

17.9 Example of a Basic Step Counter

Here is a step counter that generates sessions automatically every 60 minutes. It reports steps, distance, activity time, and calories burned. It uses relative time and has a CMOS that has a relative time counter that is initialized on the hour. It runs continuously. To make this example simpler the zero time is 24 hours before the first upload.

After connecting, doing service discovery, pairing, the collector reads the AM peripheral’s Current Time Characteristic:

```
0x00  0x80, 0x51, 0x01, 0x00
```

The current time is a relative time. The value of the current time is `0x15180` or 86,400 seconds (or 24 hours) since start assuming the counter started at 0. The collector’s current time is 6:00:00 Dec 27th 2015.

The collector then enables the characteristic descriptors for the device’s Activity Session and Count characteristics.

The AM peripheral uploads its data.

The first characteristic value indicated is the Activity Session:

```
0x00, 0x01, 0x11, 0x10, 0x0E, 0x00,       0x50, 0x46, 0x00, 0x00
```

This session characteristic indicates store and forward (`bit 0 cleared`). The session identifier is set to 1. The activity type is walking (`0x11` or 17 decimal). The session duration `0x0E10` is one hour. The time stamp is `0x4650` which is 19 hours earlier than the current time value `0x15180` read by the collector. That means this session started at 11:00:00 on Dec 26th 2015.

The next characteristic value indicated is the child Count Characteristic:

```
0x03, 0x01, 0x6A, 0x18, 0x00,       0x00, 0x88, 0x13, 0x01, 0x4A, 0x01  0xA8,
0x0C, 0x00
```

[a child Rates characteristic value indication could go here]

This characteristic value contains counts and the optional distance, calories burned, and activity time fields. Its parent identifier is session 1. The count `0x186A` (6250 decimal) gives the number of steps (unit code `0x00`) taken and the distance walked is `0x1388` (5000) meters (unit code `0x01`). The energy expended is `0x014A` (330) nutritional calories. The actual time during the session spent walking is `0x0CA8` or 54 minutes.

The next characteristic value indicated is a second Activity session:

```
0x00, 0x02, 0x11, 0x10, 0x0E, 0x00,       0x60, 0x54, 0x00, 0x00
```

which is still a walk, and it contains an activity time, has identifier 2, has a duration of one hour, and a start time of `0x5460` which is 18 hours before the current time of `0x15180` seconds thus 12:00:00 on Dec 26th 2015.

The next characteristic value indicated is the child Count Characteristic:

```
0x03, 0x02, 0xA5, 0x00, 0x00,       0x00, 0x84, 0x00, 0x01, 0x0A, 0x00, 0xB4,
0x00, 0x00,
```

[a child Rates characteristic value indication could go here]

which for this session shows 165 steps for a total of 132 meters, 10 calories burned and an activity time of 180 seconds or three minutes.

17.10 Activity Monitor Service

The AM Service requires:

- Sensor Current Time
- Activity Session
- Count Characteristic (store-and-forward only)
- Toggle Characteristic

It may optionally contain:

- Rates
- Acceleration
- Configuration Characteristic

17.11 Activity types

The different types of activities are reported as codes. They will have one of the following values:

Activity	code

MDC_HF_ACT_AMB	0
MDC_HF_ACT_REST	1
MDC_HF_ACT_MOTOR	2
MDC_HF_ACT_LYING	3
MDC_HF_ACT_SLEEP	4
MDC_HF_ACT_PHYS	5
MDC_HF_ACT_SUS_PHYS	6
MDC_HF_ACT_UNKNOWN	7
MDC_HF_ACT_MULTIPLE	8
MDC_HF_ACT_MONITOR	9
MDC_HF_ACT_SKI	10
MDC_HF_ACT_RUN	11
MDC_HF_ACT_BIKE	12
MDC_HF_ACT_STAIR	13
MDC_HF_ACT_ROW	14
MDC_HF_ACT_HOME	15
MDC_HF_ACT_WORK	16
MDC_HF_ACT_WALK	17
MDC_HF_ACT_EXERCISE_BIKE	18
MDC_HF_ACT_GOLF	19
MDC_HF_ACT_HIKE	20
MDC_HF_ACT_SWIM	21
MDC_HF_ACT_AEROBICS	22
MDC_HF_ACT_DUMBBELL	23
MDC_HF_ACT_WEIGHT	24
MDC_HF_ACT_BAND	25
MDC_HF_ACT_STRETCH	26
MDC_HF_ACT_YOGA	27
MDC_HF_ACT_WATER_WALK	28

17.12 Sensor Current Time Characteristic (required)

This characteristic allows the collector to obtain the current time of the Activity Monitor. A flag in this characteristic indicates the supported time type. What type of time supported is indicated in this characteristic will be used in all characteristics that report time stamps. If the AM wants to have its current time set by the collector, it indicates so by making the characteristic writable and setting a flag that requests the collector to set the time. If the AM requests it's time to be set in this manner, the collector shall set the time to the base offset or absolute time format depending upon the time type supported.

Read required (write optional but required if the time is to be set).

17.12.1 Flags

1 byte.

(1) **0: relative time used (1)**

Set when using relative time. A relative time stamp is used when the device wants to let the collector handle the mapping to wall clock time. This time stamp allows the device to report a counter that could start at zero (unless the device asks the collector to set the time in which case the relative time will start at the set value of the seconds), perhaps at the factory or when a new battery is put in, and monotonically advances until the battery is depleted. The current value can be read by the collector such that the collector then can map all reported time stamps to a wall clock time based on the differences between this value and the values reported in the measurements. The relative time units are in the same units as the base time (seconds).

(2) **1: base offset time used (1)**

Set when using base offset time.

(3) **2: absolute time used (1)**

Set when using absolute time.

(4) **3: Daylight Savings Time (DST) supported (1)**

If set this indicates that the time stamps know about DST. Always cleared when the time stamp is relative

(5) **4: DST (1)**

If set it indicates that daylight savings applies**.** Always cleared when the time stamp is relative or daylight savings time is not supported.

(6) **5: Collector set time (1)**

If set it indicates that the AM peripheral requires it's time to be set by the collector. The collector shall set the time if this flag is set within 10 seconds of reading this characteristic. The collector always sets the time to its current time. The AM peripheral shall adjust the time stamp of any stored data to the new time line.

17.12.2 Current Time

4-7 bytes.

This field contains the current time in seconds plus offset if using Base Offset time. A special value of all `0xFF` in elements is used to indicate the time is not known (for example the collector or user has yet to set the time). This value indicates the collector shall set the time. Base offset time is represented as the number of seconds since 00:00 on 1st January 2000 UTC as a 32 bit unsigned integer (without accounting for leap seconds in a similar fashion to NTP) and a 16 bit signed integer field gives the offset from base time to local time in minutes. The offset is the amount of time one **adds** to the UTC base time to get the local time. There are no fractional seconds as in the IEEE standard.

When supporting an Absolute time, you should use 7 bytes. The time is encoded in the standard manner for BTLE absolute time.

Since BTLE data is always reported little endian the Base offset fields appear as follows in the characteristic:

```
Seconds:                    offset:
      0xss, 0xss, 0xss, 0xss, 0xzz, 0xzz
      LSB               LSB
Relative time:
      0xss, 0xss, 0xss, 0xss
      LSB
Absolute time (hours are using 24-hr clock; year is 16-bit in little endian)
      0xYY, 0xYY, 0xMM, 0xDD, 0xHH, 0xmm, 0xSS
      LSB
```

17.13 Activity Session Characteristic (Required)

This characteristic is sent to indicate the start of a session.

Sent by indication.

17.13.1 Flags

1 byte.

(1) **0: summary (1) continuous monitoring start (0)**

When set it indicates that the session is a summary session using store and forward. The duration field and one and only one of the time stamps fields shall be present. When cleared it indicates that the session is using continuous monitoring. The duration field shall not be present.

(2) **1: relative time used (1)**

Set when using relative time. If data is being stored OR this is summary session, one and only one of bits 1, 2, and 3 must be set.

(3) **2: base offset time used (1)**

Set when using base offset time. If data is being stored OR this is summary session, one and only one of bits 1, 2, and 3 must be set.

(4) **3: absolute time used (1)**

Set when using absolute time. If data is being stored OR this is summary session, one and only one of bits 1, 2, and 3 must be set.

(5) **4: Time fault (1)**

This flag is set when the current time in the Current Time Characteristic is not relevant for the time stamps in these measurements. This can happen when the current time has been lost (perhaps on a battery replacement) and this data has been stored prior to the fault. If the device uses relative times, there is no way to recover what the wall clock times of the stored data may have been, but the measurements may still be of interest. If the device uses Base Offset time, there is no way for the collector to validate the **accuracy** of the times but at least there will be some indication of what the wall clock time was.

This time fault carries to all characteristics associated with this session.

17.13.2 Session identifier

1 byte.

This value identifies the session. It is used in characteristic values to identify which session the value is associated with. Session identifiers start at 0 and increment to 255 before recycling. (Do we need this?)

17.13.3 Activity type

1 byte.

This number is one of the values in the Activity Types table and is related to the values in Table 44 of the IEEE 11073 11041 Cardiovascular specialization. Add 1000 to get the MDC value.

17.13.4 Session duration

3 bytes.

This value is the duration of the session in seconds and is only present in the store and forward (summary) mode. The value reported is NOT the amount of time one engaged in the actual activity. That quantity is defined as the activity time. For example if the session was reporting a time period of 6 hours containing a hike, the session duration would be 6 hours but the actual time of hiking might have been only 4 hours. Note that the session duration must always be >= activity time. The session duration shall equal the sum of all direct child (not children of children) session durations.

17.13.5 Session time stamp

7/6/4 bytes (Absolute/Base Offset/Relative).

The type of time stamp must match the type supported by the Current Time Characteristic. The time stamp reports the start of the session.

17.14 Count Characteristic (Required in Store-and-Forward)

This characteristic is used to report the count of some activity, for example the number of steps, the number of pedal strokes on a bicycle, the number of stairs climbed, the time resting, and so on. When this characteristic is reported as continuous monitoring or in real time, the accumulated value up to that time is reported.

This characteristic also contains the distance or cumulative distance. Devices may need input such as stride length or wheel diameter to report this value.

Sent by indication.

17.14.1 Flags

1 byte.

(1) **0: distance value and units present (1)**

Set if a distance value and units are present. Both distance and distance units are present or neither is present.

(2) **1: calories expended present (1)**

Set if the (nutritional) calories burned is present.

(3) **2: relative time used (1)**

Set when using relative time. If data is being stored OR this is summary session, one and only one of bits 1, 2, and 3 must be set.

(4) **3: base offset time used (1)**

Set when using base offset time. If data is being stored OR this is summary session, one and only one of bits 1, 2, and 3 must be set.

(5) **4: absolute time used (1)**

Set when using absolute time. If data is being stored OR this is summary session, one and only one of bits 1, 2, and 3 must be set.

(6) **5: Time fault (1)**

This flag is set when the current time in the Current Time Characteristic is not relevant for the time stamps in these measurements. This can happen when the current time has been lost (perhaps on a battery replacement) and this data has been stored prior to the fault. If the device uses relative times, there is no way to recover what the wall clock times of the stored data may have been, but the measurements may still be of interest. If the device uses Base Offset time, there is no way for the collector to validate the **accuracy** of the times but at least there will be some indication of what the wall clock time was.

This flag shall be set if the session this measurement is associated with has the time fault flag set.

17.14.2 Parent session

1 byte.

Identifies the parent session this measurement belongs to.

17.14.3 Counts

3 bytes.

The number of steps, strides, pedal strokes, and so on.

17.14.4 Count units

1 byte.

One of the following values:

0 = steps/strides
1 = pedal strokes
2 = swim strokes
3 = seconds
4 = minutes
5 = hours

17.14.5 Activity time

3 bytes.

The time spent participating in the actual activity in seconds. In store and forward mode this value is the total activity time for the session while in continuous mode it is the cumulative activity time up to the point of the measurement.

17.14.6 Distance

2 bytes.

17.14.7 Distance units

1 byte:

0 = cm
1 = m
2 = deca m
3 = hekto m
4 = kilo m
5 = in
6 = feet
0xFF = unknown

17.14.8 Calories burned

2 bytes.

Values are reported in units of kilo calories (common consumer nutritional calorie measurement).

17.14.9 Time Stamp

7/6/4 bytes (Absolute/Base Offset/Relative).

The type of time stamp supported is indicated in the Current Time Characteristic. Never present in store and forward mode.

17.15 Rates Characteristic (Optional)

This characteristic is used to report rates, speeds, and items related to rates and speeds. The heart and breathing rates are also reported in this characteristic since they possess similar properties. In the store and forward mode these quantities will be reported as statistics (means, averages, maxima, minima) over the session or activity time. In continuous mode these quantities will be reported as instantaneous values. The characteristic may also contain a measure of the activity intensity in units of percent where 100% represents maximum effort. This characteristic value shall not be sent to the collector unless it reports at least one of the speed, cadence, heart rate, breathing rate, or effort quantities.

Sent by Indication.

17.15.1 Flags

2 bytes.

(1) **0: speed and units field present (1)**

Set if the speed and unit fields are present.

(2) **1: cadence and units field present (1)**

Set if the cadence and unit fields are present.

(3) **2: heart rate present (1)**

Set if the heart rate measurement is present.

(4) **3: breathing rate present (1)**

Set if the breathing rate measurement is present.

(5) **4: effort (1)**

Set if the effort measurement is present.

(6) **5: Modifier present (1)**

This flag is set when the modifier field is present and will only be set when using the store-and-forward reporting mode.

(7) **6: relative time used (1)**

Set when using relative time. If data is being stored OR this is summary session, one and only one of bits 6, 7, and 8 must be set.

(8) **7: base offset time used (1)**

Set when using base offset time. If data is being stored OR this is summary session, one and only one of bits 6, 7, and 8 must be set.

(9) **8: absolute time used (1)**

Set when using absolute time. If data is being stored OR this is summary session, one and only one of bits 6, 7, and 8 must be set.

(10) **9: Time fault (1)**

This flag is set when the current time in the Current Time Characteristic is not relevant for the time stamps in these measurements. This can happen when the current time has been lost (perhaps on a battery replacement) and this data has been stored prior to the fault. If the device uses relative times, there is no way to recover what the wall clock times of the stored data may have been, but the measurements may still be of interest. If the device uses Base Offset time, there is no way for the collector to validate the **accuracy** of the times but at least there will be some indication of what the wall clock time was. This flag shall be set if the session this measurement is associated with has the time fault flag set.

17.15.2 Parent session

1 byte.
Identifies the parent session or sub session this measurement belongs to:

17.15.3 Modifier

1 byte.
This field is only present in the store and forward case and indicates if the reported measurements are a mean, max, min, and so on, and shall have one of the following values:

0 = session mean (`MDC_HF_MEAN_NULL_INCLUDE`)
1 = activity mean (average over the actual activity time) (`MDC_HF_MEAN_NULL_EXCLUDE`)

2 = session maximum value (`MDC_HF_MAX`)
3 = session minimum (`MDC_HF_MIN`)
4 = session RMS (`MDC_HF_RMS`)

17.15.4 Speed

2 bytes.
The speed quantity times 100.

17.15.5 Speed units

1 byte.
One of the following values:

0 = m/s
1 = kph
2 = mph
3 = cm/min
4 = in/min

17.15.6 Cadence

2 bytes.
The cadence quantity time 100.

17.15.7 Cadence units

1 byte:

0 = rpm
1 = steps/min

17.15.8 Heart Rate

2 bytes.
Heart rate in units of beats per minute.

17.15.9 Breathing Rate

2 bytes.
Breathing rate in units of breaths per minute.

17.15.10 Effort

1 bytes.
Effort in percent of maximum.

17.15.11 Time Stamp

7/6/4 bytes (Absolute/Base Offset/Relative).
The type of time stamp supported is indicated in the Current Time Characteristic. Never present in store and forward mode.

17.16 Acceleration Characteristic (Optional)

This characteristic is used to report the instantaneous values of the x, y, or z components of the acceleration and is only used in the continuous monitoring mode. It is sent using notifications with no time stamp. Each measurement contains only the x, y, or z components and it may contain several such values with the period between each sample defined. The interval is specified in the configuration characteristic. The flag settings indicate which component is being reported. Only one of the x, y, and z indicators may be set in a given characteristic value notification.

The number of samples sent in a characteristic in the PDU must be the same during a connection. That number is given by the `numberOfSamples` value in the Configuration characteristic.

Sent by Notification.

17.16.1 Flags

(1) **0: x-component (1)**

Set if the acceleration is the x-component.

(2) **1: y-component (1)**

Set if the acceleration is the y-component.

(3) **2: z-component (1)**

Set if the acceleration is the z-component.

17.16.2 Parent session

1 byte.
Identifies the parent session this measurement belongs to.

17.16.3 Acceleration component:

1 byte * (`Configuration.numberOfSamples`).

The acceleration values scaled according to the algorithm shown below.

Periodic data is mapped to the SampledData data type in FHIR. The data element in this data type is also scaled. If *y*[*i*] is the *i*th entry of the actual unscaled data from the sensor, it is scaled to *x[i]* using the following relation:

$$x(i) = \frac{(y(i) - b)}{m}$$

$$y(i) = x(i)m + b$$

where

$$m = \frac{(upperRange - lowerRange)}{(upperScaled - lowerScaled)}$$

$$b = upperRange - m * upperScaled$$

upperRange= the largest value reported by the sensor
lowerRange = the lowest value reported by the sensor
upperScaled= the scaled value corresponding to the largest value reported by the sensor
lowerScaled = the scaled value corresponding to the lowest value reported by the sensor
x[*i*] = scaled data as sent in the acceleration characteristic

y[*i*] = actual data as recorded by the sensor

17.17 Configuration characteristic (Optional)

This optional characteristic provides setting values the monitor may need to compute quantities like the calories consumed. The AM may allow the collector to set these values. If sent, at least one of the fields must be present.

Read required, write is optional.

17.17.1 Flags

(1) **0: weight present (1)**

Set if the weight field is present.

(2) **1: height present (1)**

Set if the cadence field is present.

(3) **2: age is present (1)**

Set if the age field is present.

(4) **3: stride length is present (1)**

Set if the stride length field is present.

(5) **4: acceleration data present (1)**

When set it indicates the data for handling the periodic acceleration data is present.
See the acceleration characteristic for a description of the scaling.

17.17.2 Weight

2 bytes.

Weight in kilograms times 200. Keeps the value an integer.

17.17.3 Height

2 bytes.

Height in cm.

17.17.4 Age

1 byte.

Age in years.

17.17.5 Stride Length

2 bytes.

Stride length in centimeters.

17.17.6 Sample period

2 bytes.

This field indicates the time interval between samples in milliseconds. The maximum separation is therefore 65 seconds.

17.17.7 Number of Samples per PDU

1 byte.

This field indicates the number of samples sent in each PDU. The maximum size is 19. Whatever value is selected, it must be the same in every PDU for a given connection.

17.17.8 Upper Range

4 bytes.

The highest value reported by the sensor. The scale factor is in the format of a `Mder FLOAT`.

17.17.9 Lower Range

4 bytes.

The lowest value reported by the sensor. The offset is in the format of a `Mder FLOAT`.

17.17.10 Upper Scale

1 byte.

The upper scaled value **unsigned**. When the sensor reports **upper range**, the scaled value will be **upper scale**.

17.17.11 Lower Scale

1 byte.

The lower scaled value **unsigned**. When the sensor reports **lower range,** the scaled value will be lower scale.

18. References

Renesas Healthcare Applications: https://www.renesas.com/en-us/solutions/home/healthcare.html

Healthcare Meters Kit: https://www.renesas.com/en-us/solutions/home/healthcare/hckit.html

Android Mobile App for Blood Glucose Meter, Blood Pressure Cuff, and Pulse Oximeter:Health@Home PHG: https://play.google.com/store/apps/details?id=com.lampreynetworks.ahd.health.at.home

iOS Mobile Applications for Blood Glucose Meter, Blood Pressure Cuff, and Pulse Oximeter: Health@Home PHG: https://itunes.apple.com/us/app/health-home-personal-health/id1154783934

Android Mobile App for Activity Monitor is available under software section on website.

Website and Support

- Support: https://synergygallery.renesas.com/support

Technical Contact Details:

- America: https://www.renesas.com/en-us/support/contact.html
- Europe: https://www.renesas.com/en-eu/support/contact.html
- Japan: https://www.renesas.com/ja-jp/support/contact.html

Revision History

Rev.	Date	Description	
		Page	Summary
1.00	Mar 15, 2018	-	Initial release

SALES OFFICES Renesas Electronics Corporation http://www.renesas.com

Refer to "http://www.renesas.com/" for the latest and detailed information.

Renesas Electronics America Inc.
1001 Murphy Ranch Road, Milpitas, CA 95035, U.S.A.
Tel: +1-408-432-8888, Fax: +1-408-434-5351

Renesas Electronics Canada Limited
9251 Yonge Street, Suite 8309 Richmond Hill, Ontario Canada L4C 9T3
Tel: +1-905-237-2004

Renesas Electronics Europe Limited
Dukes Meadow, Millboard Road, Bourne End, Buckinghamshire, SL8 5FH, U.K
Tel: +44-1628-651-700, Fax: +44-1628-651-804

Renesas Electronics Europe GmbH
Arcadiastrasse 10, 40472 Düsseldorf, Germany
Tel: +49-211-6503-0, Fax: +49-211-6503-1327

Renesas Electronics (China) Co., Ltd.
Room 1709 Quantum Plaza, No.27 ZhichunLu, Haidian District, Beijing, 100191 P. R. China
Tel: +86-10-8235-1155, Fax: +86-10-8235-7679

Renesas Electronics (Shanghai) Co., Ltd.
Unit 301, Tower A, Central Towers, 555 Langao Road, Putuo District, Shanghai, 200333 P. R. China
Tel: +86-21-2226-0888, Fax: +86-21-2226-0999

Renesas Electronics Hong Kong Limited
Unit 1601-1611, 16/F., Tower 2, Grand Century Place, 193 Prince Edward Road West, Mongkok, Kowloon, Hong Kong
Tel: +852-2265-6688, Fax: +852 2886-9022

Renesas Electronics Taiwan Co., Ltd.
13F, No. 363, Fu Shing North Road, Taipei 10543, Taiwan
Tel: +886-2-8175-9600, Fax: +886 2-8175-9670

Renesas Electronics Singapore Pte. Ltd.
80 Bendemeer Road, Unit #06-02 Hyflux Innovation Centre, Singapore 339949
Tel: +65-6213-0200, Fax: +65-6213-0300

Renesas Electronics Malaysia Sdn.Bhd.
Unit 1207, Block B, Menara Amcorp, Amcorp Trade Centre, No. 18, Jln Persiaran Barat, 46050 Petaling Jaya, Selangor Darul Ehsan, Malaysia
Tel: +60-3-7955-9390, Fax: +60-3-7955-9510

Renesas Electronics India Pvt. Ltd.
No.777C, 100 Feet Road, HAL 2nd Stage, Indiranagar, Bangalore 560 038, India
Tel: +91-80-67208700, Fax: +91-80-67208777

Renesas Electronics Korea Co., Ltd.
17F, KAMCO Yangjae Tower, 262, Gangnam-daero, Gangnam-gu, Seoul, 06265 Korea
Tel: +82-2-558-3737, Fax: +82-2-558-5338

Colophon 7.0